DTIC FILE COPY

(2)

the Optical Sciences Company

P.O. Box 1329, Placentia, California 92670 ■ Phone (714) 524-3622

Report No. DR-514

Final Report on ONR Contract No. N00014-88-C-0692
Phased Array Imaging

David L. Fried
and
Douglas T. Sherwood

the Optical Sciences Company
380 S. Placentia, Suite J
Placentia, CA 92670

September 1990

Final Report for Period: 1 September 88–30 April 90

Prepared for: Office of Naval Research
Department of The Navy
800 N. Quincy Street
Arlington, Va 22217-5000

the
Optical
Sciences
Company

P.O. Box 1329, Placentia, California 92670 ■ Phone (714) 524-3622

Report No. DR-514

Final Report on ONR Contract No. N00014-88-C-0692
Phased Array Imaging

David L. Fried
and
Douglas T. Sherwood

the Optical Sciences Company
380 S. Placentia, Suite J
Placentia, CA 92670

September 1990

Final Report for Period: 1 September 88–30 April 90

Prepared for: Office of Naval Research
Department of The Navy
800 N. Quincy Street
Arlington, Va 22217-5000

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) DR-514	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION the Optical Sciences Company 6b. OFFICE SYMBOL (If applicable) 9D674	7a. NAME OF MONITORING ORGANIZATION Department of the Navy Office of Naval Research
6c. ADDRESS (City, State, and ZIP Code) P. O. Box 1329 Placentia, CA 92670	7b. ADDRESS (City, State, and ZIP Code) 800 N. Quincy Street Arlington, VA 22217-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Office of Naval Research 8b. OFFICE SYMBOL (If applicable) N00014	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-88-C-0692
8c. ADDRESS (City, State, and ZIP Code) 800 N. Quincy Street Arlington, VA 22217-5000	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Final Report on ONR Contract N00014-88-C-0692, Phased Array Imaging

12. PERSONAL AUTHOR(S) David L. Fried, Douglas T. Sherwood

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM 880901 TO 900430	9009	

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Super resolution, sparse array, CLEAN, image resolution, noise, image processing. (RH)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

can

The problem of recoverable image resolution is investigated for the case where an imaging array is used which array has an optical transfer function that may be described as consisting of "islands" of nonzero value in a sea of zero values. ~~We wish to know if~~ Can the missing spatial frequency information can be provided--~~if~~, in effect, a form of (interpolative) super resolution ~~can be achieved~~. The CLEAN algorithm used by radio astronomers suggests that this should be possible. The results developed here indicate that this can be done, with no significant price in terms of signal-to-noise ratio to be paid, and further show that a nonlinear algorithm, like CLEAN, is not required. The results show that the feasibility of doing this depends on the angular size of the object being imaged. We find that its size must be less than the inverse of the largest gap between "islands" in the array's optical transfer function. Keywords; —

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Marilyn E. Kruger	22b. TELEPHONE (Include Area Code) 714-524-3622 22c. OFFICE SYMBOL 9D674

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted.
All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

Table of Contents

ABSTRACT

The problem of recoverable image resolution is investigated for the case where an imaging array is used which array has an optical transfer function that may be described as consisting of "islands" of nonzero value in a sea of zero values. We wish to know if the missing spatial frequency information can be provided—if, in effect, a form of (interpolative) super resolution can be achieved. The CLEAN algorithm used by radio astronomers suggests that this should be possible. The results developed here indicate that this can be done, with no significant price in terms of signal-to-noise ratio to be paid, and further show that a nonlinear algorithm, like CLEAN, is not required. The results show that the feasibility of doing this depends on the angular size of the object being imaged. We find that its size must be less than the inverse of the largest gap between "islands" in the array's optical transfer function.

Chapter 1

Introduction and Summary

1.1. Introduction and Summary

The objective of the work reported here has its origin in the desire to form a high resolution image of a satellite in a geosynchronous orbit with ground based optical imaging equipment. Underlying the willingness to express such a desire is the presumption that by the exploitation of some technique or combination of techniques—drawn from the methodologies of adaptive optics, of white light speckle imagery, and of other possible relevant approaches—it will be possible to circumvent the seeming limits to resolution that are imposed by atmospheric turbulence. Once we grant such a presumption we next encounter the problem posed by the shear physical size of the instrument required for the task.

The nominal range involved is $R = 4 \times 10^7$ m. If the imaging is to be carried out utilizing $\lambda = 5 \times 10^{-7}$ m wavelength light, and the desired linear resolution on the satellite is $\delta x = 0.1$ m, then the span, $S_{\max}$ of the imaging system's aperture must be

$$\begin{aligned} S_{\max} &= R\lambda/\delta x \\ &= (4 \times 10^7)(5 \times 10^{-7})/(0.1) \\ &= 200 \text{ m}, \end{aligned} \tag{1.1}$$

This dimension is much too great for us to be able to even toy with the idea that imaging system might be a conventional sort of "full aperture" imaging telescope. Clearly the imaging system design has to be based on some sort of array concept—and a sparse array concept at that.

Once we accepted the position that the instrument design should be based on a sparse array concept, the next matter for consideration has to do with just how sparse an array we might be able to use. The kind of instrument and its cost would be strongly influenced by the degree of sparsity we could tolerate.

We had initially maintained that since the resolution we sought, δx, corresponded to a spatial frequency, $\kappa_{\max}$, such that

$$\kappa_{\max} = S_{\max}/\lambda, \tag{1.2}$$

then we would need an array pattern which provided for "coverage" of all spatial frequencies, κ, such that

$$|\kappa| \leq \kappa_{\max}. \tag{1.3}$$

By the term "coverage" we meant that the array's optical transfer function for that spatial frequency, $\tau(\kappa)$, has a non-zero magnitude. This in turn means that for any spatial frequency, κ, satisfying Eq. (1.3), there must be at least one pair of points in the "composite aperture" associated with the entire array which points are separated by a distance S, where

$$\mathrm{S} = \kappa\lambda. \tag{1.4}$$

Put differently, the requirement is that for all S such that

$$|\mathrm{S}| \leq S_{\max}, \tag{1.5}$$

there must be at least one pair of points in the array's composite aperture whose separation is equal to S. If such a pair of points do not exist then for the spatial frequency $\kappa = \mathrm{S}/\lambda$, the optical transfer function, $\tau(\kappa)$, of the array will be zero. This would seem to imply that any information about the satellite that is represented in its image by this spatial frequency, κ, would be missing in the image eventually developed from the array's measurements.

The consequence of this is that the minimum acceptable size of the individual elements of the array is "tied" to the spacing of the array elements. If for example, the spacing of array elements is in integer multiples of 2 m, then the array elements must have diameters in excess of 1 m. If they do not, then there will be spatial frequencies for which $\mathrm{S} = \kappa\lambda$ has a magnitude just larger

than 1 m, for which the optical transfer function would be zero—and this, we argued, would result in images of little utility—strongly "defaced" by the absence of significant portions of the spatial frequency spectrum. The implications of this seemed to be two-fold. First, a great deal of physical aperture area (i.e., a great deal of glass) would have to be provided. Second, unless we were prepared to consider a very large number of separate array elements, the aperture diameter of the individual array element would have to be considerably larger than the effective coherence diameter, r_0, imposed by atmospheric turbulence considerations. This carried with it the implication that just as we probably would have to be able to phase the array elements to each other, so also we would probably have to provide a wavefront distortion compensation/accommodation procedure for each array element's aperture.

It was at this point in the consideration of this matter that Ken Johnston of NRL introduced a quite different position. He argued that the radio astronomer's experience with the CLEAN algorithm demonstrated that it was possible to recover very acceptable, high resolution images starting with data from an array so sparse that for many of the spatial frequencies that were present in the recovered image, the value of the array's optical transfer function was equal to zero. It was not necessarily clear how/why this could be the case, but it nonetheless appeared to be true. The practical implication of this was that with reasonably large spacing between array elements, and thus with only a moderate number of such elements, it would be possible to use a quite small aperture diameter for each element—providing of course that light gathering requirements did not force the use of larger aperture diameters. In fact from a missing spatial frequencies point of view, the individual element's aperture diameter could be small enough so that there would not be any significant wavefront distortion within the element's aperture. This would close out the question of whether or not any adaptive optics were unavoidably needed to compensate for wavefront distortion in each aperture element—leaving the system design with the only unavoidable turbulence/phase compensation problem being that of how to handle to relative phase difference between the individual array elements. And of course, the permitted smallness of the individual array elements offered an advantage in terms of cost and mechanical complexity.

But before we could accept all of these advantages it was necessary to understand if the CLEAN algorithm does actually work as well as it seems to and also how/why it does so. We had to consider the possibility that CLEAN works as well as it does for the radio astronomers only because it has available to it an immense signal-to-noise ratio in the starting date. The available publications do not clearly rule out this possibility. If it did rely on a very large signal-to-noise ratio it would be of little potential utility to us in designing a sparse optical array for imaging a geosynchronous range satellite. Also we had to understand just how sparse the array could be and still produce useful images, i.e., how far apart from each other could the array elements be (or how few array elements we needed to use). It was to investigate these questions that the work reported here was undertaken.

This work is reported in the various chapters following this introduction. The balance of this introduction explains how we viewed/approached this problem, the methods we used in carry out our investigations, and finally what we found. As a convenience to the reader we will first take up a discussion of our work in a bit more detail.

To sum up, we found that it is possible to recover a quite satisfactory image even though the array is so sparse that the value of its optical transfer function is equal to zero over much of the spatial frequency domain covered by the image eventually produced. We found that this was possible even with quite modest signal-to-noise ratio values for the basic data produced by the array; we found that the image recovery process seems to impose little penalty in terms of signal-to-noise ratio. Moreover, we found that there was no strict requirement that the image recovery process be nonlinear—even though the CLEAN algorithm itself is nonlinear. We found that the size of the individual aperture elements did not matter (except as that effects the total light gathering/shot noise) but that what was of critical importance was the relationship between the size of the object being imaged and the spacing between array elements. The spacing between array elements set a critical angular dimension and the angular size of the object being imaged had to be smaller

than that if a satisfactory image was to be recovered without imposing a severe signal-to-noise ratio penalty.

Our conceptual approach to this problem was through the idea of super resolution. More than two decades ago there was a flurry of activity around the concept of super resolution—a term which can be taken as meaning, at that time, the possibility of obtaining an image containing valid data for spatial frequency whose magnitude is above D/λ (cycles/rad f.o.v) when collecting the optical information using an imaging system whose aperture diameter is D and when working in a (narrow) spectral band characterized by a wavelength λ. The value of the optical transfer function is zero for such large spatial frequencies, and the ability to obtain valid image data at such spatial frequencies with such an instrument is in a rather obvious sense "super resolution". It seemed to us that, as a practical matter, the term "super resolution" referred not so much to the magnitude of the spatial frequency but more so to the fact that the value of the optical transfer function was equal to zero for the spatial frequencies being considered. In that sense what was being accomplished by the CLEAN algorithm working with data from a very sparse array was super resolution. The spatial frequencies in question were not, in general, as large as the largest covered by the array's span, but the value of the array's optical transfer function was equal to zero for the spatial frequencies in question—and yet valid image data was being produced by the CLEAN algorithm for these spatial frequencies.

With this putative equating of the sparse array problem to the concept of super resolution, we were able to draw on the insight provided by Jim Harris more than 25 years ago, that to the extent that super resolution works, it works because the object being imaged is of finite size, on a uniform (black) background.* He showed that no two distinctly different objects of *finite angular size* can have identical images. This caused us to recognize the potential significance of the finite size of the satellite we were interested in imaging. But Harris' work was followed by a number of others that showed that while super resolution could be achieved—in fact could be achieved by quite linear signal processing methods—there was a tremendous signal-to-noise ratio penalty to be paid for what was achieved.

Based on consideration of these various results we formulated an approach to investigating what could be accomplished by processing sparse array data, which approach restricted attention to finite size objects and to linear signal processing algorithms (at least initially). We chose to use a spatially quantized (i.e., a sample value) formulation of the problem, rather than one based on the use of analytic functions. (This assured us that, though we might burden the computer, the most complex analytic operations we would have to consider were matrix manipulations.) We assumed as a given, the matrix representing the imaging system's (i.e., the array's) point spread function—a matrix which when multiplied by the data array (of sample values) representing the object, would result in a data set (of sample values) representing the raw or initial image formed by the imaging system/array. We assumed that there would be some set of random (noise) values added to this initial image formed by the imaging system/array. (We always considered a gaussian noise with an rms level that was the same for all pixels in the initial image.) We then posed our analysis in terms of a linear process (i.e., an image processing matrix) to be applied to (i.e., to be multiplied by) the data set constituting the initial image so as to produce a set of estimated values for the data set representing the object being imaged.

When we assumed that we knew nothing of the statistics of the object being imaged then we imposed a least square error criteria in the selection of our linear process (i.e., in the selection of our image processing matrix). Our least square criteria was that the recovered object (image), if imaged by the array (i.e., if multiplied by the array's optical transfer function matrix) would produce a set of data matching the row (initial) image values as closely as possible, in a least square sense. This lead in a direct manner from knowledge of the array's optical transfer function to formulation of a matrix for linear processing, (i.e. matrix multiplication) of the raw (initial) image data to produce the enhanced image. It allowed us to develop results for the mean square error in the recovered

* J. L. Harris, "Diffraction and Resolving Power" J. Opt. Soc. Am. 54 931-936 (1964).

image as a function of the rms noise and the nominal signal level in the raw image.

We also considered the case in which we assumed that we knew something about the statistics of the object being imaged. In this case we were able to use a minimum variance formulation of the image processing problem. In this case also we obtained an image enhancement matrix to be multiplied by the raw (initial) image data produced by the array, and thereby produce an enhanced image. In this case also we developed an expression for the mean square error in the recovered image as a function of the mean square error in the raw image and of the nominal signal level in the raw image.

In developing this line of investigation we recognized that it might not be desirable to attempt to recover an image containing all spatial frequencies up to the highest that the sample density (or pixel size) could support. Accordingly we considered spatial frequency analysis of the recovered image. We developed expressions allowing us to calculate the signal-to-noise ratio to be associated with each spatial frequency component of the recovered/enhanced image, and the signal-to-nose ratio to be associated with an image enhanced for all spatial frequencies up to some cut-off frequency, and with zero content for higher spatial frequencies. Such results were developed for both minimum variance and least square error formulations.

Our initial work started with a two-aperture one-dimensional array, and with a three-aperture two-dimensional array. This work along with our basic analytic formulation of the problem is presented in Chapters 2 and 3 of this report. Unfortunately we were able to develop virtually no useful insight from this work, and the documentation is included here partly for completeness, but mostly as a reference for the detail of the analytic formulation—which we continued to use. We now realize that our failure to obtain useful insight from the results of this work has to do with the fact that the problem was not "rich" enough. The two-aperture one-dimensional array and the three-aperture two-dimensional arrays are too simple. In subsequent work we considered multi-aperture one-dimensional arrays. For these multi-aperture arrays the imaging process/problem was sufficiently rich that we were able to develop considerable insight into the image enhancement/super resolution process. (We did not pursue the two-dimensional array problem any further, considering that the insight we got from the one dimensional array problem was equally applicable to the two-dimensional array.) After a brief discussion of the one-dimensional array results that we obtained with linear processing we will discuss work done with nonlinear processing algorithm; such results are developed in Chapter 4.

As an instructive example of the results we obtained with a linear processing image enhancement technique we present the minimum variance results obtained with the one-dimension six aperture (non redundant) sparse array whose array pattern and optical transfer function are shown in Fig. 1.1. We have a sparse array span denoted by D, with an optical transfer function that has non-zero values out to a spatial frequency of D/λ. The optical transfer function has islands of nonzero value with spacing of $\frac{1}{17}D/\lambda$ cycles/rad$_{\text{f.o.v.}}$ in spatial frequency—out to $\frac{13}{17}D/\lambda$ cycles/rad$_{\text{f.o.v.}}$, but then has a gap of $\frac{3}{17}D/\lambda$ cycles/rad$_{\text{f.o.v.}}$ to the next island. Between the islands, in the gaps of $\frac{1}{17}D/\lambda$ and $\frac{3}{17}D/\lambda$, the value of the optical transfer function is zero. We call particular attention to this transition from a gap of $\frac{1}{17}D/\lambda$ to $\frac{3}{17}D/\lambda$ cycles/rad$_{\text{f.o.v.}}$ at a spatial frequency of $\frac{13}{17}D\lambda$ cycles/rad$_{\text{f.o.v.}}$. We also note that the value of the optical transfer function is zero for all spatial frequencies above D/λ cycles/rad$_{\text{f.o.v.}}$.

The analysis associated with this array was formulated utilizing pixel size of $\frac{1}{4}\lambda/D$ both to define the object and its recovered image, and to represent the (noisy) raw (initial) image formed by the array. With this sample size we are able to consider spatial frequencies up to $2D/\lambda$. We took as our knowledge of the object being imaged, the assumption that it was a zero-mean white noise pattern, i.e., that there was no correlation between pixel values, that the average value of each pixel was zero, and that the standard deviation in the value of each pixel was the same—a value denoted by σ_x. We considered the noise in the array's image plane to also be zero mean white noise with a standard deviation denoted by σ_n. We relate σ_x and σ_n by presume a scale factor in associating numerical values with the raw image data such that the matrix representing the array's

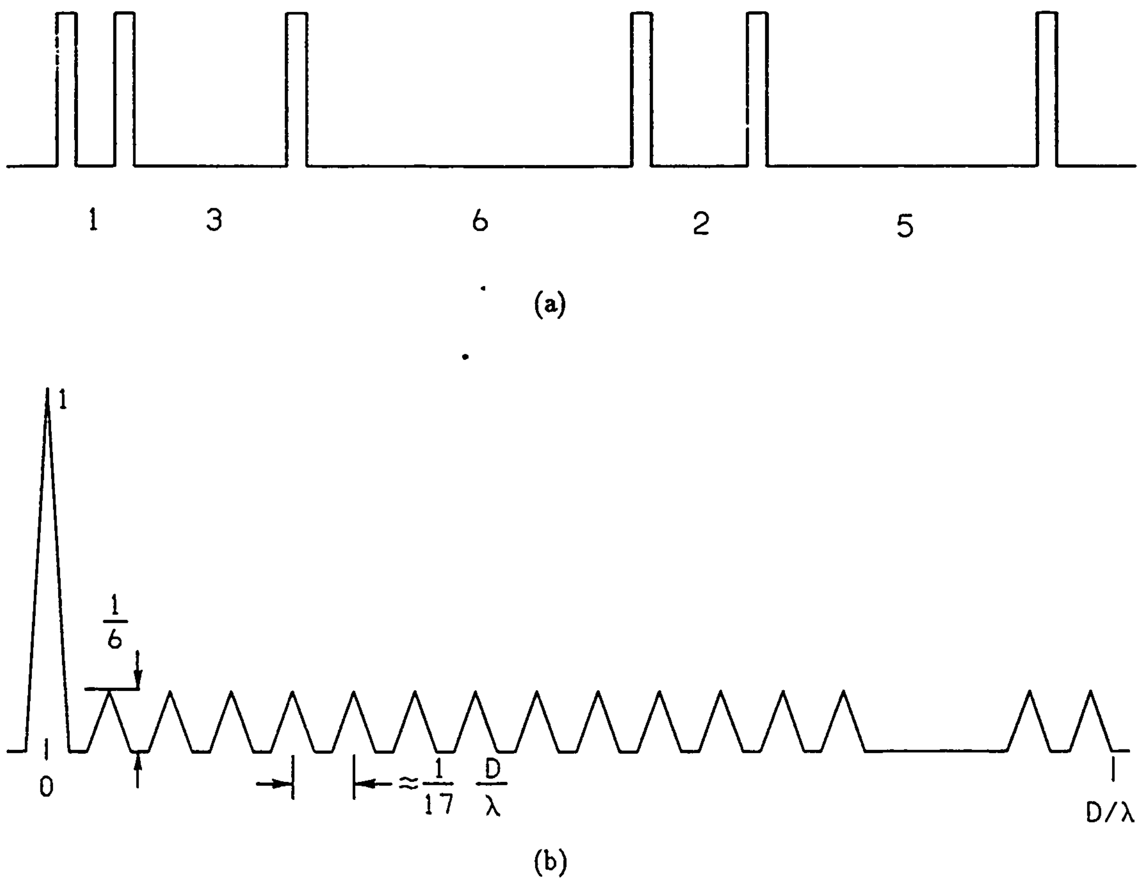


Figure 1.1 Array Pattern and Optical Transfer Function for a Six-Element One-Dimensional Non-Redundant Array. The spacing of the array elements, indicated by the numbers 1, 3, 6, 2, and 5 in Fig. 1.1a total to 17, which we associate with the array span, to be denoted by D. It is to be understood as significant that the array elements are small enough compared to the array spacing that the value of the optical transfer function is equal to zero over much of the spatial frequency domain of interest. We see in Fig. 1.1b that the space between the individual "islands" of nonzero OTF value is $\frac{1}{17}D/\lambda$, out to $\frac{13}{17}D/\lambda$, with only small zero-value regions between the islands—but there is a significantly larger gap of zero-value OTF between the $\frac{13}{17}$ D/λ spatial frequency island and the $\frac{16}{17}D/\lambda$ island, a gap that will be of considerable significance in interpreting our results.

point spread function would image an everywhere equal to unity infinite extent object as an every equal to unity infinite extent raw image. For our analysis we considered the ease where the raw data "signal-to-noise ratio" was,

$$\mathrm{SNR}_{\mathrm{RAW}} = 20\ \log_{10}(\sigma_x/\sigma_n), \tag{1.6}$$

equal to 50.

When we calculated mean square error in each spatial frequency component of the processed image produced by the minimum variance estimator for various size objects, we obtained the results shown in Fig. 1.2. These results are for object sizes of $3\lambda/D$, $6\lambda/D$, $9\lambda/D$, ..., $21\lambda/D$. Examining this figure we can see that only for the two largest objects, $18\lambda/D$ and $21\lambda/D$, do we make much error in estimating the spatial frequency components for *all* spatial frequencies below $\frac{13}{17}D/\lambda$, even for spatial frequencies for which the arrays optical transfer function is equal to zero. For spatial frequencies between $\frac{13}{17}D/\lambda$ and $\frac{16}{17}D\lambda$, it is only for the two smallest objects, $3\lambda/D$ and $6\lambda/D$, that acceptable results are obtained. The quality of the estimate for spatial frequencies beyond D/λ falls off quite rapidly, going out the farthest for the smallest objects.

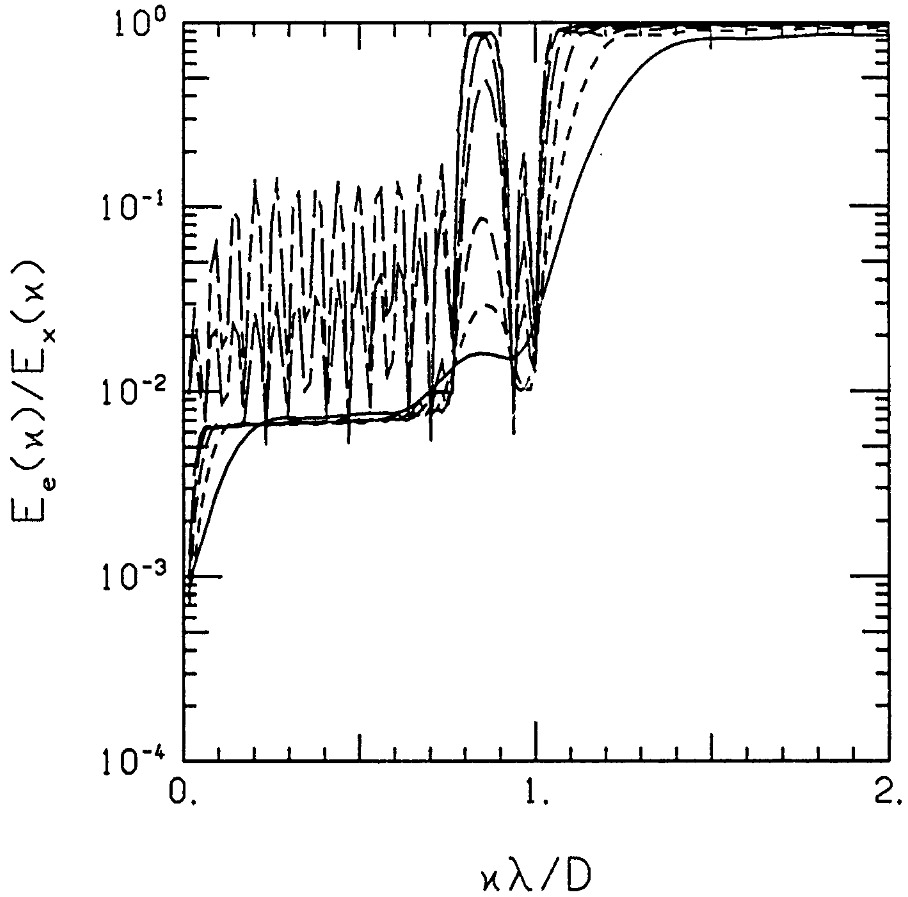


Figure 1.2. Spatial Frequency Component Estimation Error.

Results are shown here for the six-element sparse array depicted in Fig. 1.1, for object of size $3\lambda/D$, $6\lambda/D$, $9\lambda/D$, ..., $21\lambda/D$. Each curve represents the mean square error, $E_e(\kappa)$, in our estimate of a spatial frequency component, κ, normalized with respect to the mean square value expected for that component, $E_x(\kappa)$. Minimum variance estimation with $\frac{1}{4}\lambda/D$ sized pixels and an initial signal-to-noise ratio, $\mathrm{SNR}_{\mathrm{RAW}}$, of 50 dB is utilized in developing these results. It should be noted in studying the results depicted here, that only for the $18\lambda/D$ and $21\lambda/D$ cases do the curves show any " excess" noise for spatial frequencies below $\frac{13}{17}D/\lambda$—even for those spatial frequencies for which the MTF, as shown in Fig. 1.1b, is equal to zero.

In Fig. 1.3 we show related results, only for all object sizes from $1\lambda/D$, $2\lambda/D$, ..., to $32\lambda/D$, and in terms of the rms error in the enhanced image. The results are shown as a function of spatial frequency cut-off, κ_0, in the enhanced image. The signal to noise ratio is defined by the equation

$$\mathrm{SNR}_{\mathrm{EST}} = 20\ \log_{10}(\sigma_{\hat{x}}/\sigma_n), \tag{1.7}$$

where $\sigma_{\hat{x}}$ denotes the standard deviation in the pixel values of the enhanced image. The results shown in Fig. 1.3 may be considered as being obtained as a spatial frequency integral over (an extended set of) the results shown in Fig. 1.2.

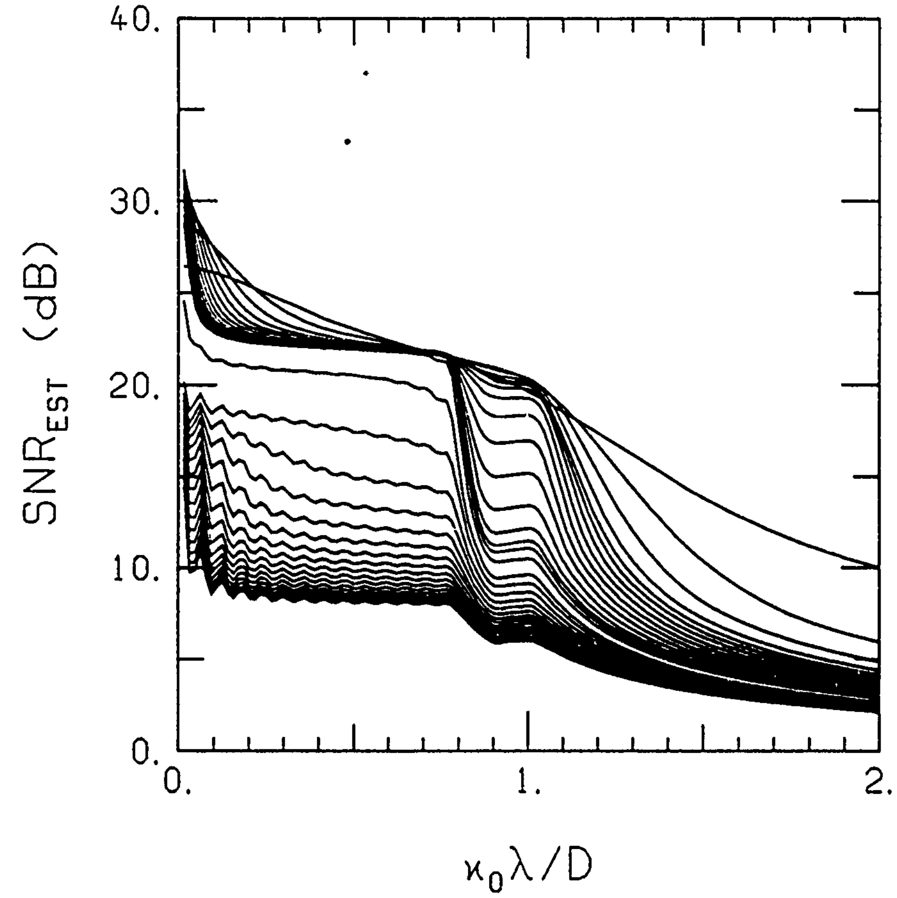


Figure 1.3. Enhanced Image Signal-to-Noise Ratio as a Function of Spatial Frequency Cut-off. The results shown here may be thought of as being developed from those in Fig. 1.2 (except that here we consider object sizes of $1\lambda/D$, $2\lambda/D$, $3\lambda/D$, ..., $32\lambda/D$), by integrating over spatial frequency up to the cut-off frequency, κ_0. These results correspond to the mean square error in the estimated pixel value of the enhanced image, normalized by the variance (associated with the object's statistics). It is to be noted that good results are obtained for object sizes up to $17\lambda/D$ (the inverse in the basic optical transfer function gap shown in Fig. 1.1b, for spatial frequency cut-offs up to $\frac{13}{17}D/\lambda$. For cut-offs up to D/λ good results are obtained only for object sizes about up to $6\lambda/D$, about equal to the inverse of the largest gap in the optical transfer function when considering spatial frequencies up to D/λ.

It is clear from an examination of these results that there is a break in enhanced image quality at a spatial frequency of D/λ and an earlier break at $\frac{13}{17}D/\lambda$ for all objects except those whose size is greater than $6\lambda/D$. Most important of all, we note that performance is seriously degraded for any spatial frequency cut-off for objects larger than $17D/\lambda$..

All of these results suggest to us that the critical consideration is the size of the object relative to the inverse in the largest gap between "islands" in the optical transfer function. If we take $\frac{13}{17}D/\lambda$ as our cut-off spatial frequency, then the largest gap is $\frac{1}{17}D/\lambda$—and we get good results for all objects whose size is no greater than $17\lambda/D$. This is made quite explicit by the presentation of the data shown in Fig. 1.4. If we take D/λ as our cut-off spatial frequency, then the largest gap is $\frac{13}{17}D/\lambda = \frac{1}{5.67}D/\lambda$, and the only good results are for objects whose size is no greater than $6\lambda/D$. We conclude from these results and others like it, that super resolution, i.e., the ability to develop image data for spatial frequencies for which the value of the imaging system's optical transfer function is zero, can be obtained by linear processing of the raw image data. The (only) requirement is that the object be smaller than the inverse of the largest gap in the modulation transfer function, and that we formulate our image processing algorithm with this knowledge in hand.

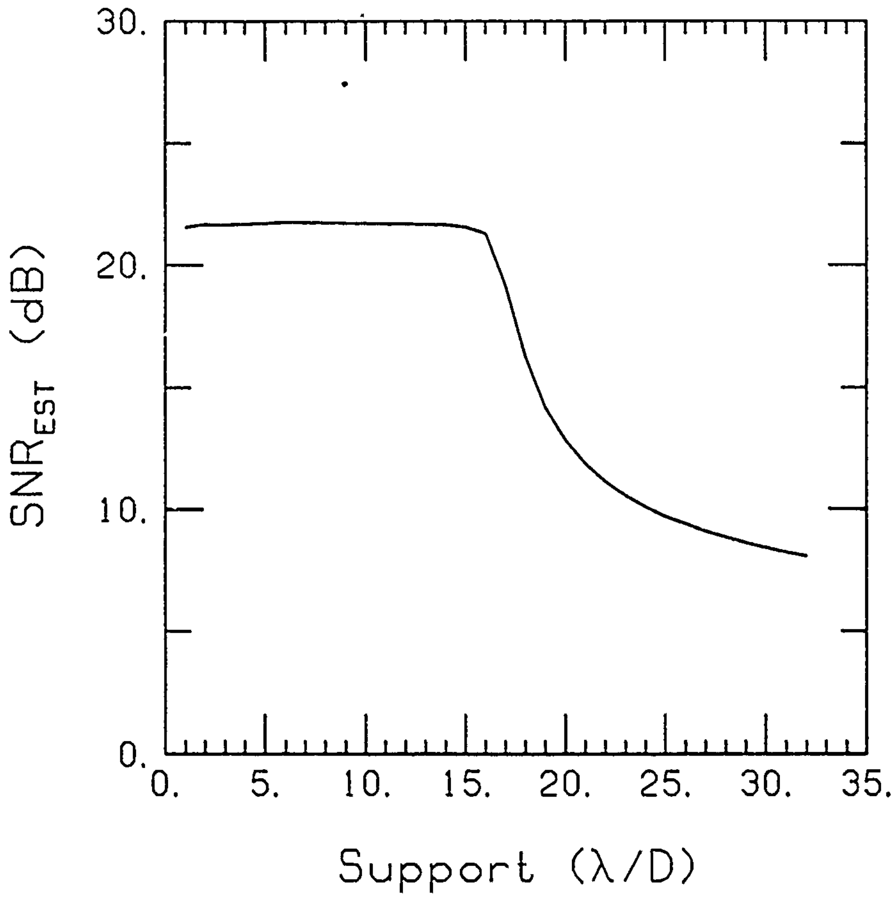


Figure 1.4. Enhanced Image Signal-to-Noise Ratio as a Function of Object Size. **The results shown here are taken directly from Fig. 1.3, for a cut-off spatial frequency of $\frac{13}{17}D/\lambda$. For this cut-off spatial frequency the largest gap between islands of nonzero value in the optical transfer function shown in Fig. 1.1b, is $\frac{1}{17}D/\lambda$. We consider it significant that the signal-to-noise ratio does not fall-off, i.e., apparently super resolution has been achieved so long as the object size is no greater than the inverse of this gap.**

To understand why it should be that we can apparently develop the missing data for spatial frequencies for which the optical transfer function is zero—in essence achieving super resolution—we undertook the study reported in Chapter 5. In this work we show that if we start with an infinite extent white noise pattern—a pattern for which there is zero correlation between the amplitudes of the components at two distinct spatial frequencies, no matter how close the spatial frequencies—and

if we slice out a limited size sample of this random pattern and embed it in an infinite extent of zero values, then we now have an infinite random pattern for which there is non-zero correlation between different spatial frequency components! However, only if the components are for spatial frequency components that are close enough together is the correlation substantial. We find that the concept of spatial frequencies that are close enough together corresponds to the inverse of the difference of the two frequencies being greater than the size of the random sample that we extracted from the original random pattern.

We interpret this as meaning that in any finite support random pattern, spatial frequency components are necessarily correlated if the spatial frequencies are close enough, where close enough has to do with the inverse of the objects size. We believe that the minimum variance estimator naturally takes advantage of this correlation. This allows it to estimate the amplitude of the unmeasured spatial frequency components, using the amplitude of the measured components as a basis for the estimates—providing that the object is not too large to allow accurate estimation to be done this way. Put in a different way, we would say that the correlation indicates that the missing spatial frequencies do not contain any significant amount of new information; it's just a matter of "spreading out" the information we do get from the measurement in a somewhat different way, and this is what the algorithm does. It's still "super resolution" according to our definition of that term as providing image data for spatial frequencies for which the optical transfer function is equal to zero—but somehow the wonder of it, of *super* resolution seems to be gone. Nonetheless, we shall call it "super resolution".

It may be recalled that the CLEAN algorithm and other related algorithms possibly promising super resolution were nonlinear in nature. We have thus far restricted our attention to linear algorithms. This would seem to leave open the possibility that if we had used a suitable nonlinear algorithm we would have been able to process larger objects. To investigate this possibility, since the nonlinear algorithms do not lend themselves to closed form analytic treatment, as were used in Chapter 4, we switched to a Monte Carlo approach. We developed results using a Monte Carlo approach, and then compared those results (i.e., the rms errors) with the rms error for a linear algorithm—namely the least square algorithm. These results are presented in Chapter 6.

We carried out Monte Carlo runs for both the CLEAN algorithm and for a least square augmented by positivity requirement. In general we got essentially the same performance from the nonlinear as from the linear algorithm. This is illustrated in Fig. 1.5. Based on this and the other results developed in Chapter 6 we have concluded that there is no special virtue in the use of a nonlinear image enhancement algorithm—though there may be practical computational advantages—and that for analysis of expected performance we may rely on results developed with a linear algorithm being used for image enhancement.

But most important of all, we have concluded that Ken Johnston's suggestion is valid—that we could use an imaging array with widely spaced array elements and small element size to image an object in a geosynchronous orbit. Exactly how small an aperture diameter each array element can have is no doubt going to be strongly influenced by light gathering/signal strength considerations, and it may be possible to use aperture diameters small enough to avoid wavefront distortion. Certainly we now know that there is no reason to rule out the use of such small apertures on the basis of the consequent presence of zeros in the array's optical transfer function.

Regarding the spacing between array elements we note that if we were interested in imaging a 10 m diameter satellite in geosynchronous orbit, working at 0.5 μm wavelength, then the basic array element spacing parameter would be 2.0 m. If light gathering considerations did not preclude it there is no reason why the size of the array elements could not be as small as 10 cm, or less.

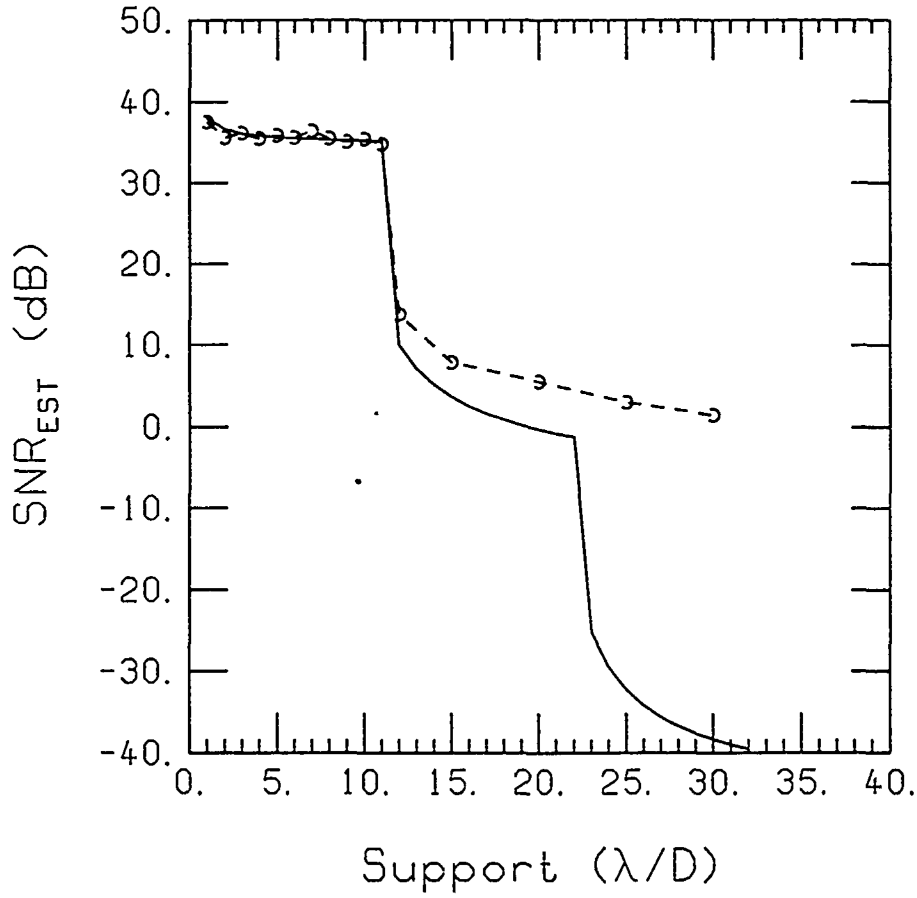


Figure 1.5. Comparison of Image Enhancement Results Using Least Square Estimation and Using CLEAN. The continuous curve represents the analytic results obtained for least square estimation. The dots (connected by the broken line) represent the results obtained from 40 Monte Carlo runs for each dot, for the CLEAN algorithm. It is obvious from consideration of these results that there is no significant advantage obtained by use of a nonlinear algorithm, i.e., CLEAN, rather than the linear image enhancement algorithm associated with least square error processing. (For more details regarding the results shown here the readers attention is directed to Fig. 6.33 and the associated discussion in Chapter 6.)

Chapter 2

Arrays of Optical Apertures: A Preliminary Investigation

(originally issued as TR-1012)

2.1. Introduction

This chapter presents the results of a preliminary investigation into the feasibility of obtaining useful images of objects viewed through a sparse array of optical apertures. If an array of apertures is sufficiently sparse, then the modulation transfer function (MTF) of the array will contain holes, or regions of zero response. Thus, the Fourier transform of a noise-free image of an object will also have corresponding regions of zero frequency response. To obtain a useful estimate of the object intensity distribution from the image, one needs to "fill in" the missing frequency regions. We refer to this process as (interpolative) super-resolution*. To do so, one must have constraint information about the object. For example, if one knows the support region of the object, then theoretically, knowledge of the fourier transform of the object intensity distribution in one region can be used to fill in the missing information in another region (analytic continuation). A positivity constraint on the estimate of the object intensity distribution can also be used in addition to, or instead of, the support constraint to extend object frequency information. In this report, our primary emphasis is on the object support constraint.

We begin in Section 2.2 by developing a discrete model of an optical system so that the system response can be approximated with a matrix equation.

In Section 2.3 we assume we have first and second moment information about the object intensity distribution and observation noise, and we develop a minimum-variance estimate of the object using the support constraint only. The results indicate that the achievement of useful super-resolution with significantly sparse arrays requires very large signal-to-noise ratios.

In Section 2.4 we investigate the implication of the addition of a positivity constraint to the finite-support constraint. Here we assume no statistical knowledge and use a least-squares estimator instead. Unfortunately, the nonlinear nature of the positivity constraint greatly complicates the computation of useful object estimates, and our results to date are consequently rather limited. However, these results indicate that positivity, when used in addition to finite support, has the potential of significantly improving super-resolution performance.

2.2. Discrete Optical Model

The intensity distribution of a monochromatic image of a object can be described mathematically as the two-dimensional convolution of the intensity distribution of the object with the point-spread function of the optical system. Since the spatial frequency response of any optical system has finite support, the point-spread function is infinite in extent. In order to study super resolution quantitatively, we have chosen to use a discrete matrix model of convolution. This necessitates not only discretizing the optical model, but assuming that the point-spread function is of finite extent. Thus, the spatial frequency response of the model will not have finite support. However, we assume that if the extent of the point-spread function is chosen sufficiently large, truncation of the point-spread function will have a minimal effect on the results. We will use both one and two dimensional discrete models. The one-dimensional model, being computationally less complex, will enable the generation of a large body of results that are easily displayed. The two-dimensional model will be used to verify that the conclusions drawn from consideration of the one-dimensional results extend to two dimensions.

* The term "super resolution" has customarily been used to refer to the development of image/object information associated with spatial frequencies higher than those that go with the largest dimension of the aperture of the imaging system—spatial frequencies for which the imaging system's transfer function has zero value. It is entirely consistent with this to generalize the concept of "super resolution" to the development of image/object information for any spatial frequency for which the imaging system's transfer function has zero value—even if the spatial frequency is less than that associated with the largest dimension of the aperture. For a filled aperture this generalization is an empty one, but for a sparse array imaging system this generalization represents a significant "extension" of the basic concept of "super resolution". We may use the terms "interpolative super resolution" and "extrapolative super resolution" to distinguish between the case 1) where we are "filling in" missing data at spatial frequencies less than the highest the aperture can nominally provide, and 2) when we are "filling in" missing data at spatial frequencies greater than the highest the aperture can nominally provide, respectively.

2.2.1 One-Dimensional Model

Let the components of the $L \times 1$ vector $\mathbf{z}$ be the intensity pixels of the object line, let the $N \times 1$ vector $\mathbf{y}$ represent the image line intensity pixels, and let the $N \times L$ matrix B be the transformation from object line to image line (its columns contain shifted versions of the system point-spread-function). Then an image can be represented in the matrix equation

$$\mathbf{y} = B\mathbf{z} + \mathbf{n}, \tag{2.1}$$

where $\mathbf{n}$ is a $N \times 1$ noise vector. The product $B\mathbf{z}$ in (1) represents convolution of the object line with the point spread-function. Let the point spread function be $h(k)$, where $k = -K, -K+1, \ldots, 0, +1, +2, \ldots, +K$. Then the B matrix is

$$B = \begin{bmatrix} h(-K) & 0 & \ldots & 0 \\ h(-K+1) & h(-K) & \ldots & 0 \\ \vdots & h(-K+1) & \ldots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ldots & 0 \\ h(0) & \vdots & \ldots & 0 \\ \vdots & h(0) & \ldots & h(-K) \\ \vdots & \vdots & \ldots & h(-K+1) \\ h(K-1) & \vdots & \ldots & \vdots \\ h(K) & h(K-1) & \ldots & \vdots \\ 0 & h(K) & \ldots & h(0) \\ 0 & 0 & \ldots & \vdots \\ \vdots & 0 & \ldots & \vdots \\ \vdots & \vdots & \ldots & h(K-1) \\ 0 & 0 & \ldots & h(K) \end{bmatrix}. \tag{2.2}$$

Since B has N rows, from (2) we must have that $N = L + 2K$.

Now let the $M \times 1$ vector $\mathbf{x}$ represent an object of finite contiguous support of length $M \leq L$ pixels located somewhere in the object line. We can write

$$\mathbf{z} = W\mathbf{x}, \tag{2.3}$$

where the $L \times M$ matrix W is of the form

$$W = \begin{bmatrix} & & 0 & \\ \hline 1 & & & 0 \\ & 1 & & \\ & & \ddots & \\ 0 & & & 1 \\ \hline & & 0 & \end{bmatrix}. \tag{2.4}$$

Thus, W comprises an $M \times M$ identity matrix embedded in a $L \times M$ matrix of zeros. Combining Eq.'s (2.3) and (2.1) yields

$$\mathbf{y} = BW\mathbf{x} + \mathbf{n}. \tag{2.5}$$

To complete the model, we need a point-spread function. To derive a point-spread function, we first need to choose an aperture function. To represent a sparse array of apertures in one dimension we will use a simple model of two small apertures of length d with outside spacing D, as shown in Fig. 2.1. The corresponding modulation transfer function (MTF) is given below, where κ is the spatial frequency variable and λ is wavelength;

$$MTF(\kappa) = \frac{1}{A}\int dx\, w(x + \tfrac{1}{2}\kappa\lambda)w(x - \tfrac{1}{2}\kappa\lambda). \tag{2.6}$$

Normally A is chosen to be the area (in this case the length) of the aperture, so that the MTF at zero spatial frequency is unity. However, we wish to "penalize" the MTF of a sparse aperture to reflect the fact that the total energy in the image plane of a sparse array will be reduced from that of a full array in proportion to the ratio $2d/D$. We will thus set A equal to D instead of the more conventional $2d$. The resulting MTF is shown in Fig. 2.2a. Fig. 2.2b shows the MTF when $d/D = \frac{1}{2}$ (the full aperture case). We note from Fig. 2.2a that, for $d < D/3$, there is a region of zero MTF over the range $d/\lambda \leq \kappa \leq D/\lambda - 2d/\lambda$ It is this region that will bear scrutiny in performance evaluations. Fig.'s 2.3-2.7 show the MTF on a log-log scale for various values of d/D.

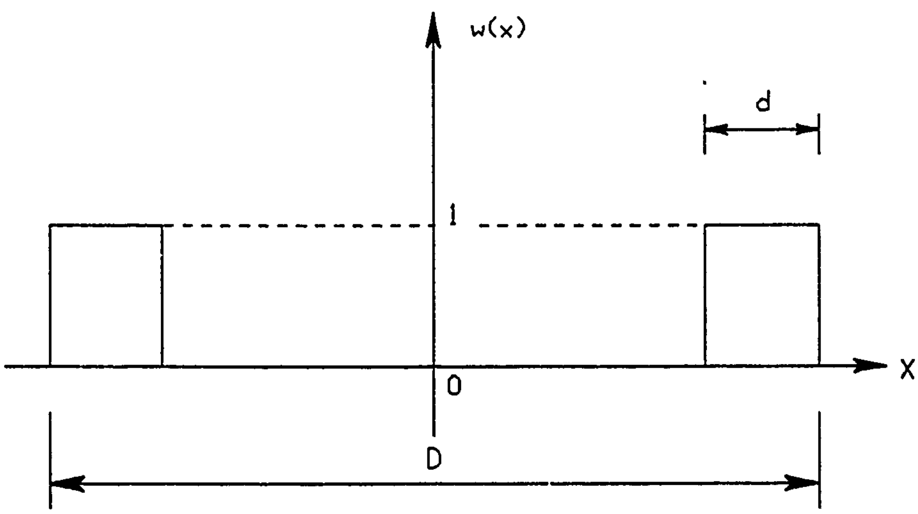
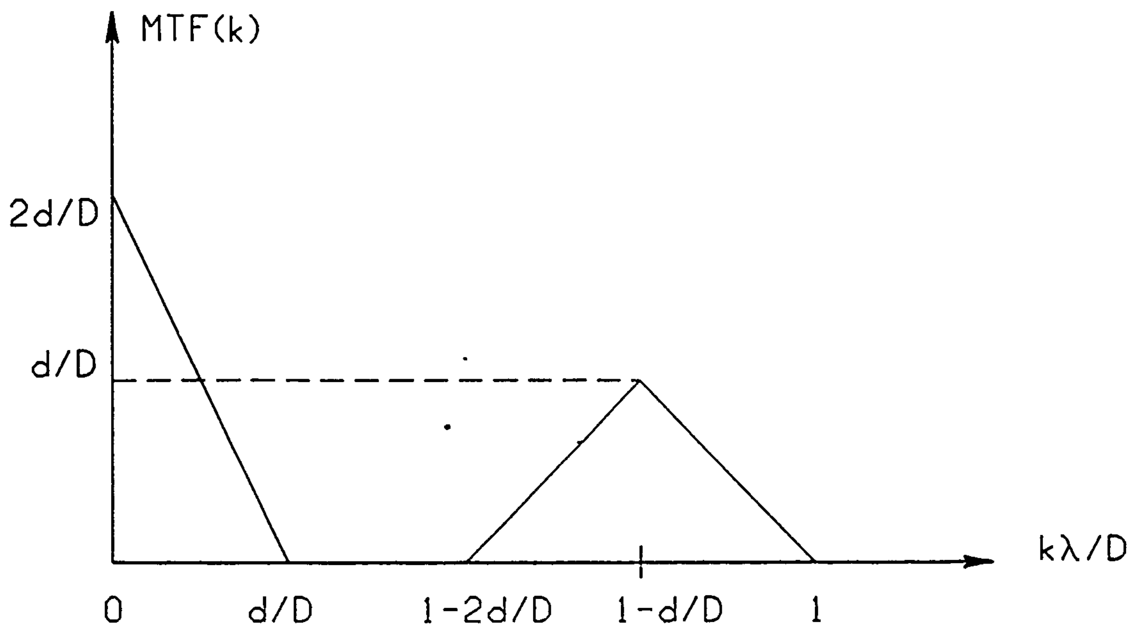


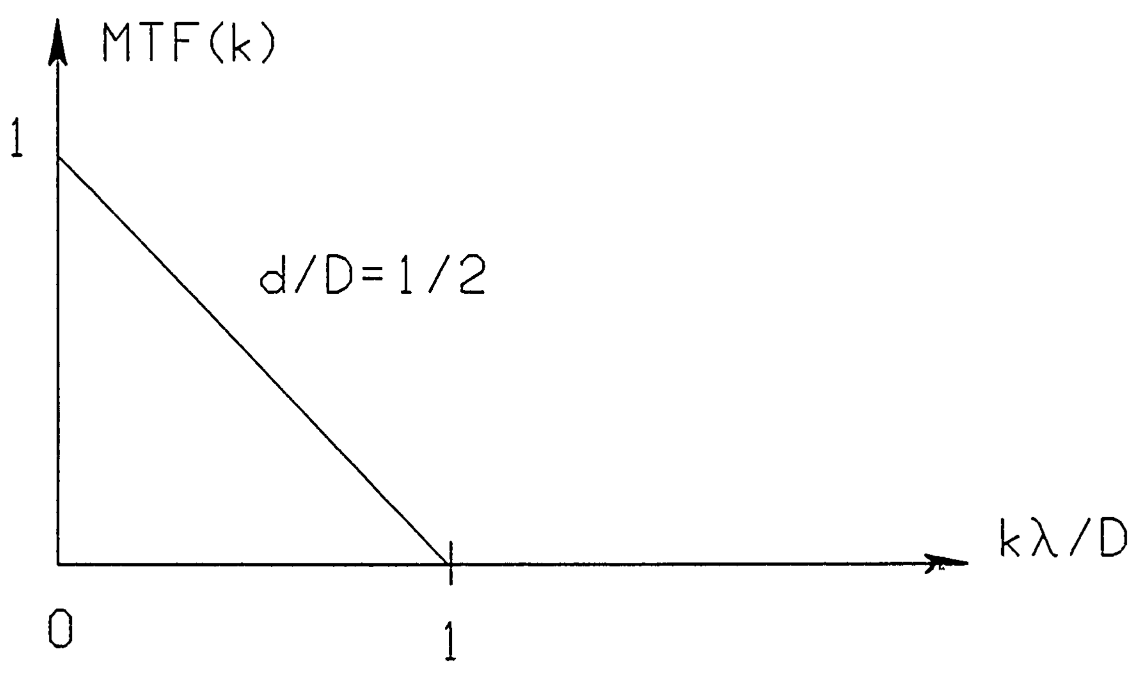
Figure 2.1. One-Dimensional Aperture Function

The point-spread function of an optical system is the inverse Fourier transform of the MTF function. Without going into the details, one can show that the inverse Fourier transform of the MTF shown in Fig. 2.2a is

$$h_c(x) = \frac{4d^2}{D\lambda}\cos^2\left[\frac{\pi(D-d)}{\lambda}x\right]\left[\frac{\sin(\pi x d/\lambda)}{\pi x d/\lambda}\right]^2. \tag{2.7}$$



(a)



(b)

Figure 2.2. Modulation Transfer Function (MTF) for Aperture Function of Fig. 2.1.

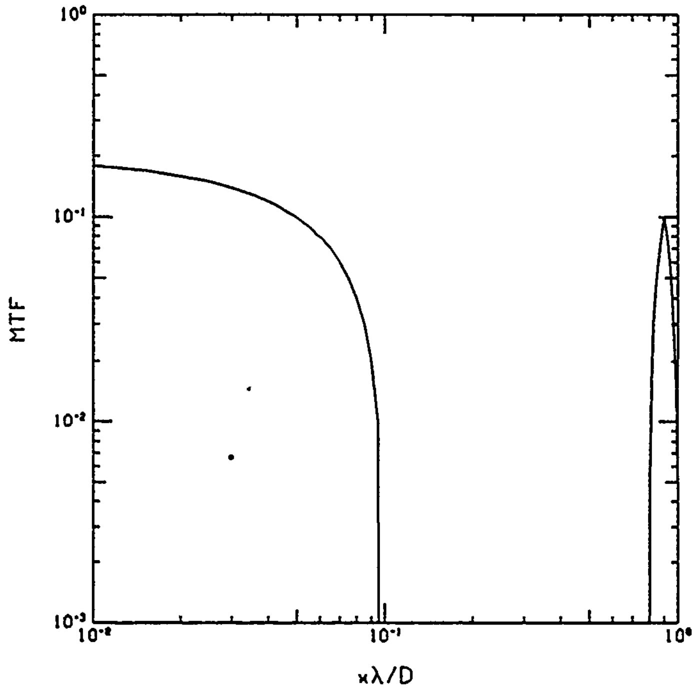


Figure 2.3. MTF of the Aperture of Fig. 2.1 with $d/D = 0.1$. The region of zero MTF is 0.7 D/λ wide.

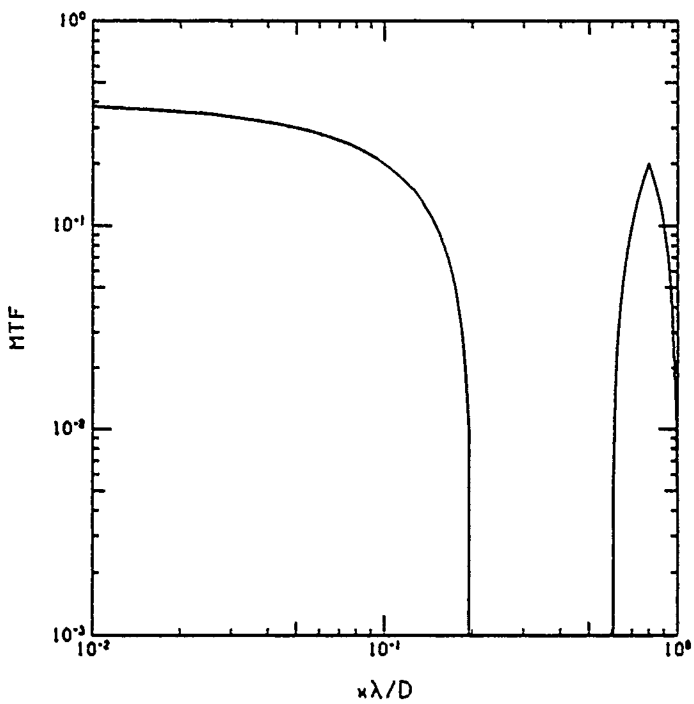


Figure 2.4. MTF of the Aperture of Fig. 2.1 with $d/D = 0.2$. The region of zero MTF is 0.4 D/λ wide.

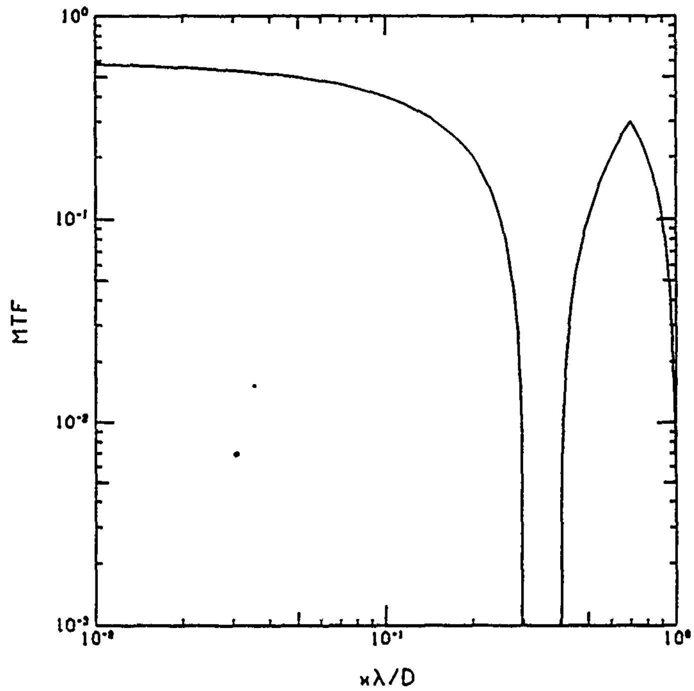


Figure 2.5. MTF of the Aperture of Fig. 2.1 with $d/D = 0.3$. The region of zero MTF is 0.1 D/λ wide.

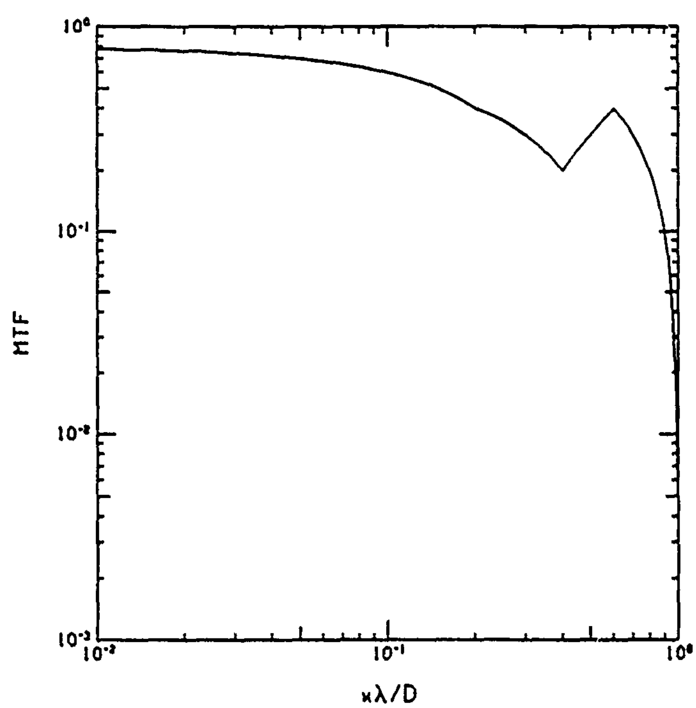


Figure 2.6. MTF of the Aperture of Fig. 2.1 with $d/D = 0.4$.

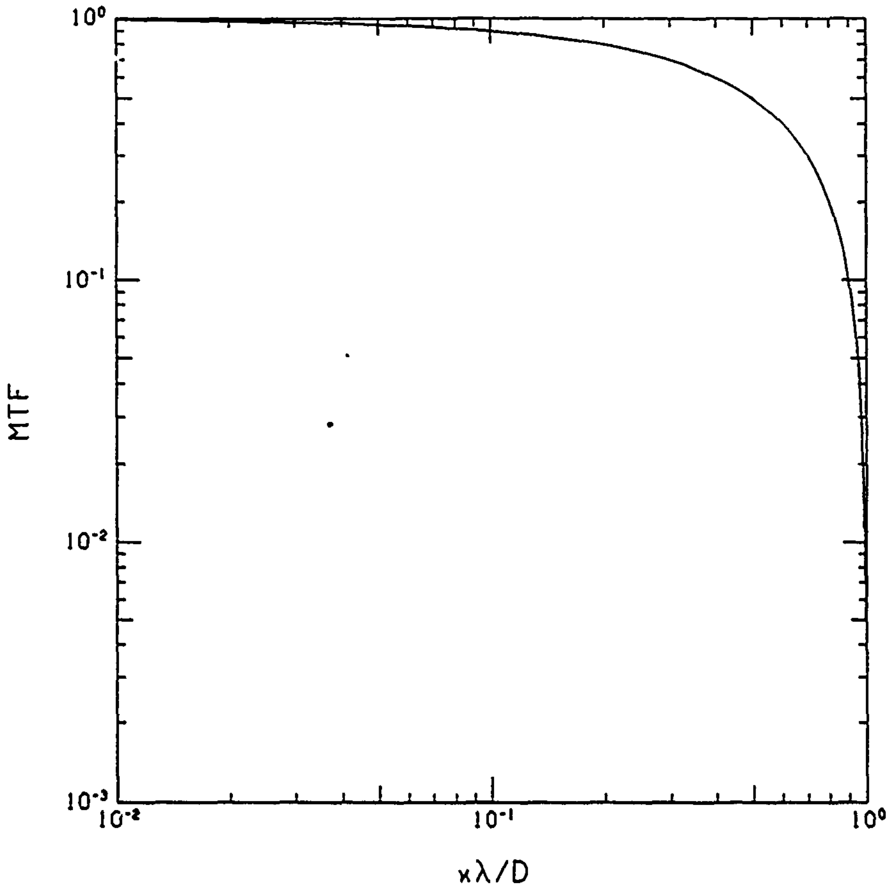


Figure 2.7. MTF of the Aperture of Fig. 2.1 with $d/D = 0.5$ (full aperture).

The subscript c denotes that this point-spread function is the continuous version. The discrete version of the point-spread function is found by sampling and truncating the continuous version. Let Δ denote the pixel spacing in the image line. Then the discrete point spread function used in the B matrix given by Eq. (2.2) is

$$h(n) = \begin{cases} \Delta h_c(n\Delta), & -K \leq n \leq K \\ 0, & \text{else.} \end{cases} \tag{2.8}$$

Before we go on to the two-dimensional model, we wish to discuss the evaluation of the performance of a processor. Any processor we chose will have, as an observable, the image vector $\mathbf{y}$. Using $\mathbf{y}$ and whatever *a priori* information is available, the processor will form an estimate of the object vector $\mathbf{x}$. Let us call this estimate $\hat{\mathbf{x}}$. Let the error in the estimate be

$$\mathbf{e} = \mathbf{x} - \hat{\mathbf{x}}. \tag{2.9}$$

One common measure of performance is the mean-square-error, $\langle \mathbf{e}^T \mathbf{e} \rangle$. For our purposes this measure is not useful. It gives us no indication of how well the processor performs as a function of spatial frequency. The performance measure we will use is the energy spectrum of the error vector. The Fourier transform of the error vector is

$$\tilde{e}(\kappa) = \sum_n e(n) \exp[-i2\pi n\Delta\kappa], \tag{2.10}$$

where e(n) is the n^{th} component of the error vector. The energy spectrum of the error vector, which we denote by $E_e(\kappa)$, is the expectation of the square of the magnitude of $\tilde{e}(\kappa)$ as given by Eq. (2.10):

$$\begin{aligned} E_e(\kappa) &= \left\langle |\tilde{e}(\kappa)|^2 \right\rangle \\ &= \sum_m \sum_n \langle e(n)e(m) \rangle \exp\left[-i2\pi\Delta\kappa(n-m)\right]. \end{aligned} \tag{2.11}$$

By plotting $E_e(\kappa)$ versus κ, we will determine how well the processor performs at each spatial frequency, thus determining its super-resolution capability. We note from Eq. (2.11) that we will need all entries of the error correlation matrix $\langle \mathbf{e}\ \mathbf{e}^T \rangle$ in the evaluation of the energy spectrum of the error, not just the diagonal terms that are required to evaluate mean-square-error. We further note that mean-square-error, $\langle \mathbf{e}^T \mathbf{e} \rangle$, is the integral of the error energy spectrum:

$$\langle \mathbf{e}^T \mathbf{e} \rangle = \Delta \int_{-\frac{1}{2\Delta}}^{\frac{1}{2\Delta}} d\kappa\, E_e(\kappa). \tag{2.12}$$

2.2.2 Two-Dimensional Model

We will generalize the one-dimensional model of the previous section to develop a two-dimensional model. We now have object and image planes, rather than lines, and a two-dimensional point spread function. Following our notation of the previous section, let the components of the vector $\mathbf{z}_n$ be the intensity pixels of the n^{th} row of the object plane, let the components of the vector $\mathbf{y}_n$ be the intensity pixels of the n^{th} row of the image plane, and let the matrix B_{nl} represent the influence that the l^{th} row of the object plane has one the n^{th} row of the image plane. Then we can write:

$$\mathbf{y}_n = \sum_l B_{nl} \mathbf{z}_l + \mathbf{n}_n, \tag{2.13}$$

where $\mathbf{n}_n$ is an additive noise vector for the n^{th} row of the image plane. Assume we truncate the point-spread function with a square window of dimension $2K+1$ on edge, and let the discrete point-spread function be $h(m,n)$, where m is the row index and n is the column index. Then the B_{nl} matrix of Eq. (2.13) has the form

$$B_{nl} = \begin{bmatrix} h(n-l,-K) & 0 & \cdots & 0 \\ h(n-l,-K+1) & h(n-l,-K) & \cdots & \vdots \\ \vdots & h(n-l,-K+1) & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ h(n-l,0) & \vdots & \cdots & 0 \\ \vdots & h(n-l,0) & \cdots & h(n-l,-K) \\ \vdots & \vdots & \cdots & h(n-l,-K+1) \\ \vdots & \vdots & \ddots & \vdots \\ h(n-l,K) & \vdots & \cdots & \vdots \\ 0 & h(n-l,K) & \cdots & \vdots \\ \vdots & 0 & \cdots & h(n-l,0) \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & h(n-l,K) \end{bmatrix}. \tag{2.14}$$

Now let us assume that the object has a contiguous region of support within the object plane. Let the components of $\mathbf{x}_l$ be the object pixels contained within the support region on the l^{th} line of the object plane. Then we can write

$$\mathbf{z}_l = W_l \mathbf{x}_l. \tag{2.15}$$

We point out that the length of $\mathbf{x}_l$ can be, in general, different for each l (some may be of zero length). The form of the W_l matrix is the same as that shown in Eq. (2.4). Substituting Eq. (2.15) into Eq. (2.13), we have,

$$\mathbf{y}_n = \sum_l B_{nl} W_l \mathbf{x}_l + \mathbf{n}_n, \tag{2.16}$$

now assume we have L rows in the object plane, N rows in the image plane, and let

$$\mathbf{y} = \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \vdots \\ \vdots \\ \mathbf{y}_N \end{bmatrix}, \tag{2.17}$$

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \vdots \\ \vdots \\ \mathbf{x}_L \end{bmatrix}, \tag{2.18}$$

$$\mathbf{n} = \begin{bmatrix} \mathbf{n}_1 \\ \mathbf{n}_2 \\ \vdots \\ \vdots \\ \vdots \\ \mathbf{n}_N \end{bmatrix}, \tag{2.19}$$

$$W = \begin{bmatrix} W_1 & & & 0 \\ & W_2 & & \\ & & \ddots & \\ 0 & & & W_L \end{bmatrix}, \tag{2.20}$$

and

$$B = \begin{bmatrix} B_{11} & B_{12} & \dots & B_{IL} \\ B_{21} & B_{22} & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ B_{N1} & \vdots & \dots & B_{NL} \end{bmatrix}, \tag{2.21}$$

Then Eq. (2.16) can be written in matrix form:

$$\mathbf{y} = BW\mathbf{x} + \mathbf{n}. \tag{2.22}$$

We need a point-spread function to complete our model. The aperture function we will use comprises three small circular apertures each of diameter d inscribed within a circle of diameter D. This aperture function is shown in Fig. 2.8. The corresponding MTF is the two-dimensional auto-correlation of the aperture function, which we can write as

$$\mathrm{MTF}(\kappa_x, \kappa_y) = \frac{1}{A} \iint dx\, dy\, w\left(x + \tfrac{1}{2}\kappa_x\lambda, y + \tfrac{1}{2}\kappa_y\lambda\right) \times w(x - \tfrac{1}{2}k_x\lambda, y - \tfrac{1}{2}k_y\lambda). \tag{2.23}$$

As in the one-dimensional model, we take A to be the area of the "full aperture", i.e. of the circumscribing circle, rather than the area of the actual aperture. We write,

$$A = \tfrac{1}{4}\pi D^2. \tag{2.24}$$

The continuous two-dimensional point-spread function is the inverse Fourier transform of the MTF:

$$h_c(x, y) = \iint d\kappa_x\, d\kappa_y\, \mathrm{MTF}(\kappa_x, \kappa_y) \exp\left[i2\pi(k_x x + k_y y)\right]. \tag{2.25}$$

Finally, the discrete point-spread function, $h(m, n)$, used in Eq. (2.14) is a sampled and truncated version of Eq. (2.25), which we can write as

$$h(m, n) = \begin{cases} \Delta^2 h_c(m\Delta, n\Delta), & -K \le m, n \le K \\ 0, & \text{else.} \end{cases} \tag{2.26}$$

Eq.'s (2.23) through (2.26) have been evaluated numerically for various values of d/D. Some examples are shown in Fig.'s 2.9 through 2.18.

To complete the model, we will compute the energy spectrum of the error. Let the estimate of the object intensity distribution be denoted by $\hat{\mathbf{x}}$:

$$\hat{\mathbf{x}} = \begin{bmatrix} \hat{\mathbf{x}}_1 \\ \hat{\mathbf{x}}_2 \\ \vdots \\ \vdots \\ \vdots \\ \hat{\mathbf{x}}_L \end{bmatrix}, \tag{2.27}$$

where $\hat{\mathbf{x}}_l$ is the estimate of the intensity distribution be denoted of the l^{th} row of the object. The error vector is

$$\begin{aligned} \mathbf{e} &= \mathbf{x} - \hat{\mathbf{x}} \\ &= \begin{bmatrix} \mathbf{e}_1 \\ \mathbf{e}_2 \\ \vdots \\ \vdots \\ \vdots \\ \mathbf{e}_L \end{bmatrix}. \end{aligned} \tag{2.28}$$

It is more convenient to express the error as a two-dimensional array of scalars and accordingly write

$$e(m,n) = (\mathbf{e}_m)_n. \tag{2.29}$$

The Fourier transform of the error array is

$$\tilde{e}(\kappa_x, \kappa_y) = \sum_m \sum_n e(m,n) \exp\left[-i2\pi(m\kappa_x + n\kappa_y)\right]. \tag{2.30}$$

The energy spectrum of the error array is

$$\begin{aligned} E_e(\kappa_x, \kappa_y) &= \left\langle |\tilde{e}(\kappa_x, \kappa_y)|^2 \right\rangle \\ &= \sum_k \sum_l \sum_m \sum_n \langle e(k,l) e(m,n) \rangle \exp\left\{i2\pi\left[(m-k)\kappa_x + (n-l)\kappa_y\right]\right\}. \end{aligned} \tag{2.31}$$

2.3. Minimum-Variance Processor

The first processor (estimator) we will investigate is the so-called minimum variance processor. We assume that we have first and second moment information on the object vector $\mathbf{x}$ and the noise vector $\mathbf{n}$ of Eq.'s (2.5) and (2.22), i.e., we know the mean vector and covariance matrices of $\mathbf{x}$ and $\mathbf{n}$. Since we know the mean values of $\mathbf{x}$ and $\mathbf{n}$, and we can compute the mean value of $\mathbf{y}$, we will assume that the mean values have been subtracted out of Eq.'s (2.5) and (2.22), and our estimate $\hat{\mathbf{x}}$ is the deviation of $\mathbf{x}$ from its mean value. In other words, we have the observation model

$$\mathbf{y} = BW\mathbf{x} + \mathbf{n}, \tag{2.32}$$

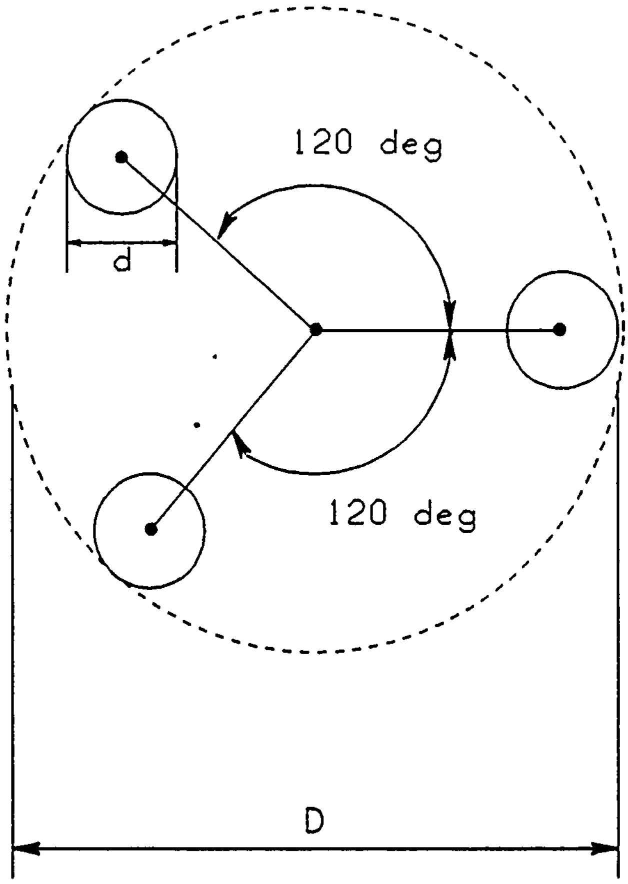


Figure 2.8. Aperture Function For Two-Dimensional Model.

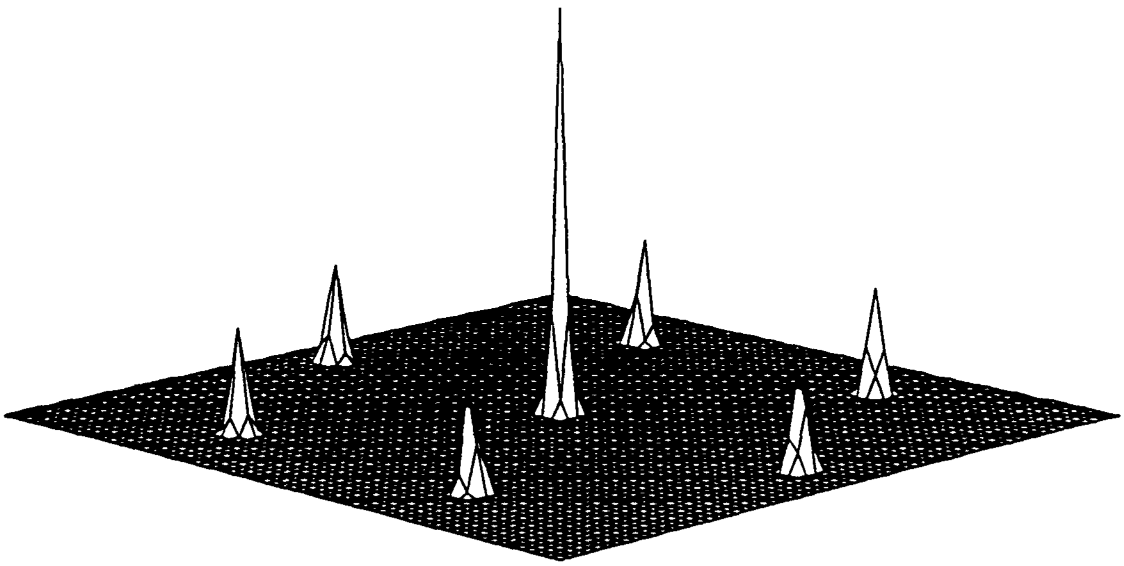


Figure 2.9. MTF of the Aperture of Fig. 2.8 with $d/D = 0.05$.
The image is 2 D/λ on edge. The height of the largest peak is $3(d/D)^2 = 0.0075$.

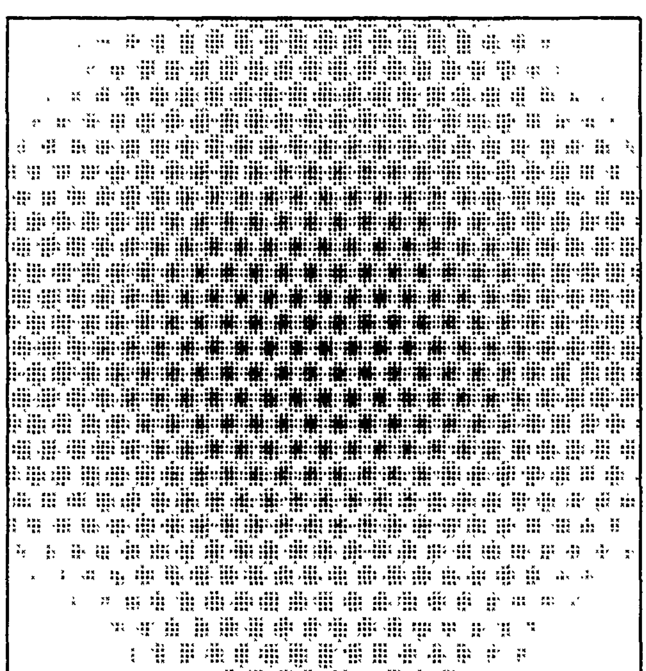


Figure 2.10. Greyscale Image of the Point-Spread Function of the Aperture of Fig. 2.8 With $d/D = 0.05$.
The image is 32 λ/D on edge.

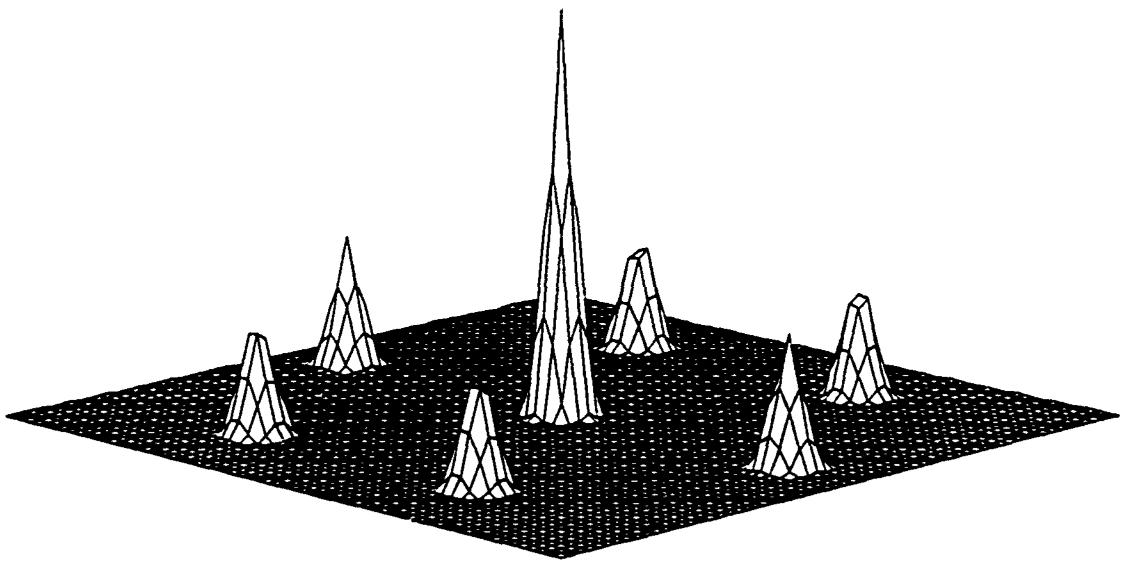


Figure 2.11. MTF of the Aperture of Fig. 2.8 With $d/D = 0.1$
The image is $2d/\lambda$ on edge. The height of the highest peak is $3(d/D)^2 = 0.03$.

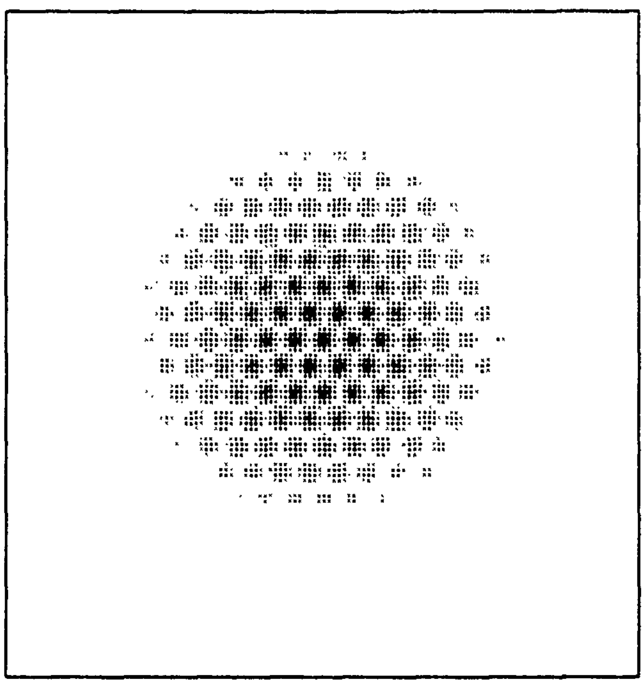


Figure 2.12. Grey-Scale Image of the Point-Spread Function of the Aperture of Fig. 2.8 With $d/D = 0.1$.
The image is 32 λ/D on edge.

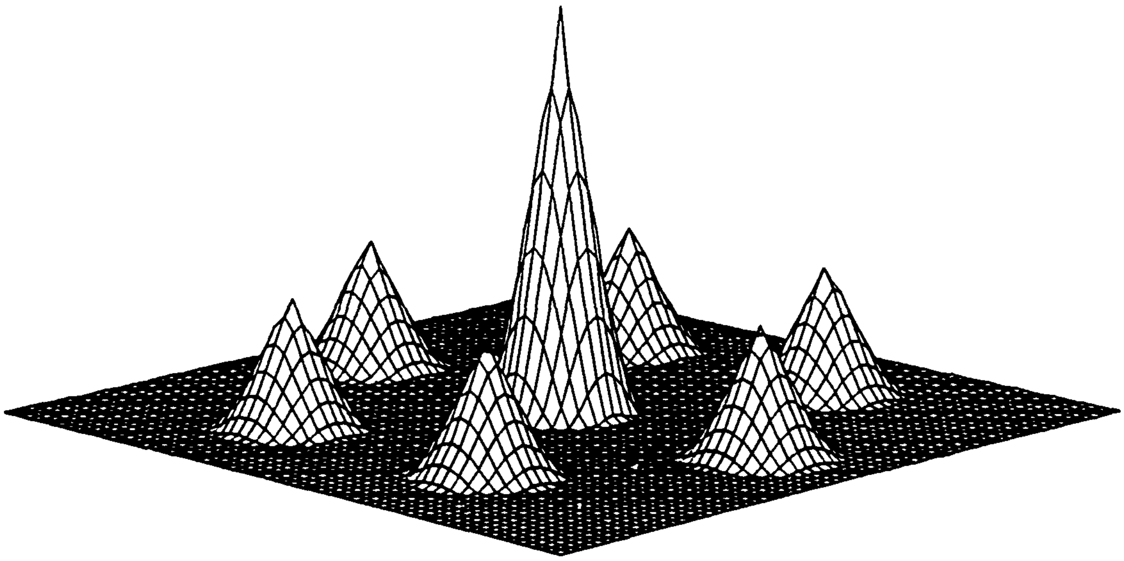


Figure 2.13. MTF of the Aperture of Fig. 2.8 With $d/D = 0.2$.
The image is $2D/\lambda$ on edge. The height of the highest peak is $3\ (d/D)^2 = 0.12$.

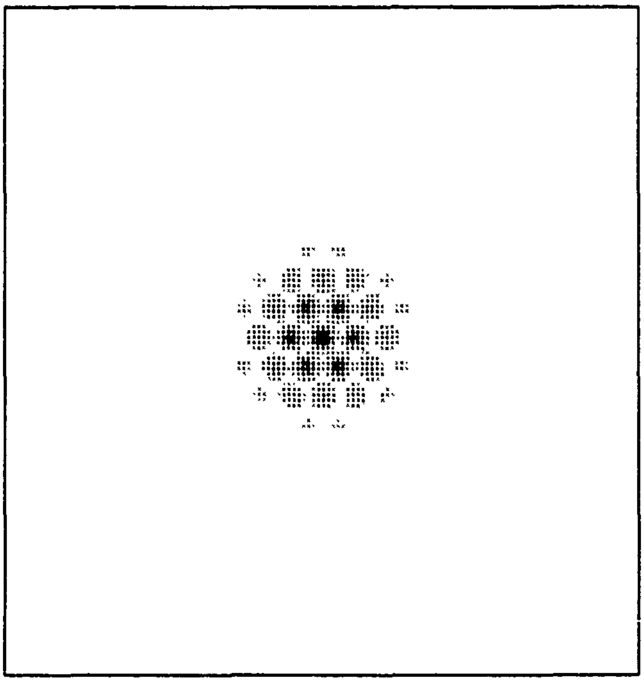


Figure 2.14. Grey-Scale Image of the Point-Spread Function of the Aperture of Fig. 2.8 With $d/D = 0.2$.
The image is $32\lambda/D$ on edge.

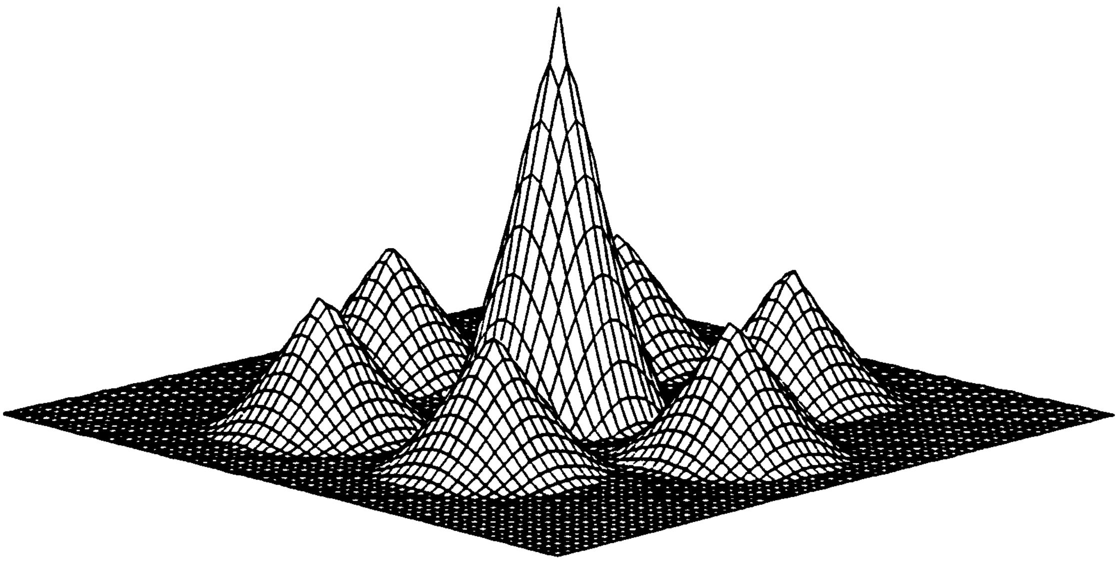


Figure 2.15. MTF of the Aperture of Fig. 2.8 With $d/D = 0.3$.
The image is $2D/\lambda$ on edge. The height of the highest peak is $3(d/D)^2 = 0.27$.

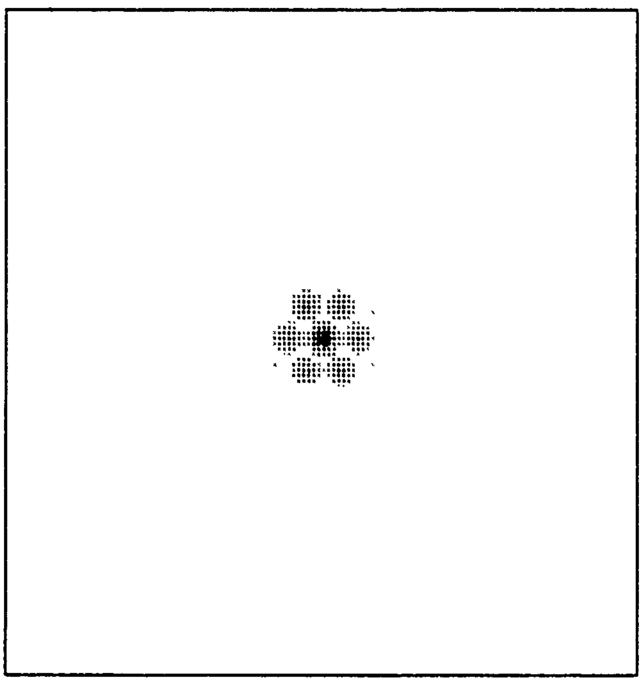


Figure 2.16. Grey-Scale Image of the Point-Spread Function of the Aperture of Fig. 2.8 With $d/D = 0.3$.
The image is $32\lambda/D$ on edge.

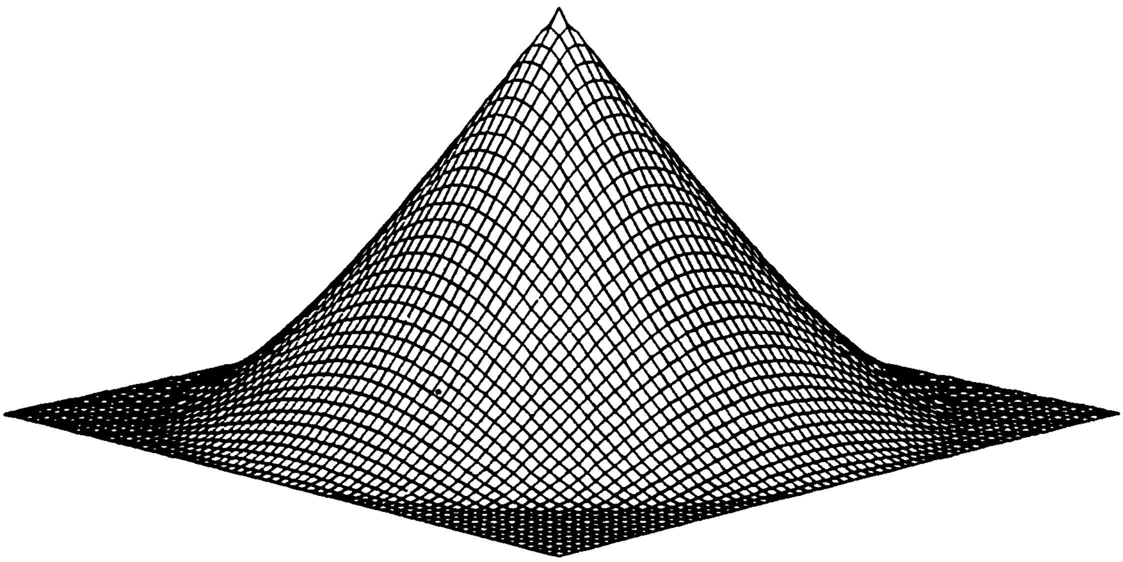


Figure 2.17. MTF of the Aperture of Fig. 2.8 With $d/D = 1$, (Full Aperture)
The image is $2D/\lambda$ on edge. The height at the peak is unity.

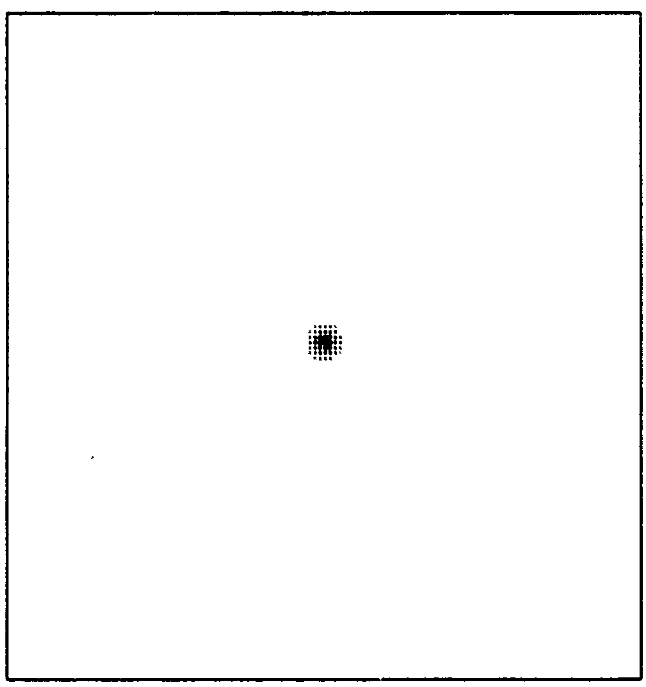


Figure 2.18. Grey-Scale Image of the Point Spread Function of the Full Aperture (Circle of Diameter D).
The image is $32\lambda/D$ on edge.

where all vectors in Eq. (2.32) have zero mean. We will use as an estimator a linear transformation of the observation vector:

$$\hat{\mathbf{x}} = H\mathbf{y}. \tag{2.33}$$

The error vector is

$$\begin{aligned} \mathbf{e} &= \mathbf{x} - \hat{\mathbf{x}} \\ &= \mathbf{x} - H\mathbf{y}. \end{aligned} \tag{2.34}$$

We wish to choose H so as to minimize the variance of the error. Since $\mathbf{e}$ is zero-mean, this is equivalent to minimizing the mean-square-error, given by

$$\begin{aligned} \xi &= \langle \mathbf{e}^T \mathbf{e} \rangle \\ &= \mathrm{Tr}\langle \mathbf{e}\mathbf{e}^T \rangle. \end{aligned} \tag{2.35}$$

Let e_n and x_n be the n^{th} components of the vectors $\mathbf{e}$ and $\mathbf{x}$, and let $\mathbf{h}_n$ be the n^{th} row of H. Then we can write:

$$e_n = x_n - \mathbf{y}^T \mathbf{h}_n. \tag{2.36}$$

We can see that choosing $\mathbf{h}_n$ to minimize $\langle e_n^2 \rangle$, for every n, minimizes ξ. To minimize $\langle e_n^2 \rangle$, we invoke the orthogonality principle (also known as the projection theorem): we choose $\mathbf{h}_n$ so that

$$\langle e_n \mathbf{y} \rangle = 0. \tag{2.37}$$

Using Eq. (2.36) in Eq. (2.37) yields the equation

$$\langle x_n \mathbf{y} \rangle = \langle \mathbf{y}\mathbf{y}^T \rangle \mathbf{h}_n. \tag{2.38}$$

Solving for $\mathbf{h}_n$ we get

$$\mathbf{h}_n = \langle \mathbf{y}\mathbf{y}^T \rangle^{-1} \langle x_n \mathbf{y} \rangle. \tag{2.39}$$

Thus, the optimum H matrix, H_o, has a value given by the expression

$$H_o = \langle \mathbf{x}\mathbf{y}^T \rangle \langle \mathbf{y}\mathbf{y}^T \rangle^{-1}. \tag{2.40}$$

The covariance matrices in Eq. (2.40) can be computed using Eq. (2.32), yielding

$$\langle \mathbf{x}\mathbf{y}^T \rangle = \langle \mathbf{x}\mathbf{x}^T \rangle (BW)^T, \tag{2.41}$$

$$\langle \mathbf{y}\mathbf{y}^T \rangle = (BW)\langle \mathbf{x}\mathbf{x}^T \rangle (BW)^T + \langle \mathbf{n}\mathbf{n}^T \rangle. \tag{2.42}$$

In computing Eq. (2.41) and (2.42) , it was assumed that this object and noise vector are uncorrelated. Using Eq. (2.41) and (2.42) in Eq. (2.40) yields

$$\begin{aligned} H_o &= \langle \mathbf{x}\mathbf{x}^T \rangle (BW)^T \left[(BW)\langle \mathbf{x}\mathbf{x}^T \rangle (BW)^T + \langle \mathbf{n}\mathbf{n}^T \rangle \right]^{-1} \\ &= R_{xx} G^T \left[G\, R_{xx} G^T + R_{nn} \right]^{-1}, \end{aligned} \tag{2.43}$$

where

$$R_{xx} = \langle \mathbf{x}\mathbf{x}^T \rangle, \tag{2.44}$$

$$R_{nn} = \langle \mathbf{n}\mathbf{n}^T \rangle, \tag{2.45}$$

$$G = BW. \tag{2.46}$$

One can apply a matrix inversion lemma to Eq. (2.43) to show that

$$H_o = (G^T R_{nn}^{-1} G + R_{xx}^{-1})^{-1} G^T R_{nn}^{-1}. \tag{2.47}$$

In order to compute the energy spectrum of the error, we need the error covariance matrix. Using Eq. (2.34) we can write

$$\langle \mathbf{e}\mathbf{e}^T \rangle = \langle \mathbf{e}\mathbf{x}^T \rangle - \langle \mathbf{e}\mathbf{y}^T \rangle H_o^T. \tag{2.48}$$

Since we have chosen H_o so that each component of the error vector is orthogonal to the observation vector $\mathbf{y}$, the second term in Eq. (2.48) must be zero. Therefore, we have

$$\begin{aligned}\langle \mathbf{e}\mathbf{e}^T \rangle &= \langle \mathbf{e}\mathbf{x}^T \rangle \\ &= \langle \mathbf{x}\mathbf{x}^T \rangle - H_o \langle \mathbf{y}\mathbf{x}^T \rangle.\end{aligned} \tag{2.49}$$

Using Eq.'s (2.41) and (2.44)–(2.47) in Eq. (2.49) yields

$$\begin{aligned}\langle \mathbf{e}\mathbf{e}^T \rangle &= R_{xx} - \left(G^T R_{nn}^{-1} G + R_{xx}^{-1}\right)^{-1} G^T R_{nn}^{-1} G R_{xx} \\ &= \left(G^T R_{nn}^{-1} G + R_{xx}^{-1}\right)^{-1} \left[\left(G^T R_{nn}^{-1} G + R_{xx}^{-1}\right) R_{xx} - G^T R_{nn}^{-1} G R_{xx}\right] \\ &= \left(G^T R_{nn}^{-1} G + R_{xx}^{-1}\right)^{-1}.\end{aligned} \tag{2.50}$$

To simplify computation of Eq. (2.50), we will assume that both the object and noise vectors are white, i.e.,

$$R_{xx} = \sigma_x^2 I, \tag{2.51}$$

$$R_{nn} = \sigma_n^2 I, \tag{2.52}$$

where σ_x^2 and σ_n^2 are the variances of object and noise pixels, respectively. If we define a signal-to-noise ratio as

$$\mathrm{SNR} = \sigma_x^2/\sigma_n^2, \tag{2.53}$$

then Eq. (2.50) can be written

$$\langle \mathbf{e}\mathbf{e}^T \rangle = (G^T G(\mathrm{SNR}) + I)^{-1} \sigma_x^2. \tag{2.54}$$

The numerator of the SNR defined by Eq. (2.53) is referenced to the object plane and the denominator is referenced to the image plane. Can we relate this to a similarly-defined SNR totally referenced to the image plane? In general, the answer is no. Such an SNR would vary from pixel to pixel. Rather than attempting to develop a suitable definition for SNR in the image plane, we will instead simply make the following observation. Given a very large object of uniform intensity, the intensity of the image is also uniform and is equal to the intensity of the object scaled by the MTF evaluated at zero spatial frequency. One can show that the scale factor for the one-dimensional model is $2d/D$ and for the two-dimensional model is $3(d/D)^2$ (evaluate Eq. (2.6) and (2.23), respectively, at zero spatial frequency).

2.3.1 Minimum-Variance Results for the One-Dimensional Case.

In order to proceed, we must choose several parameters. Two of the parameters that strongly interact are pixel spacing Δ and the length of the point-spread function $(2K+1)\Delta$. Since the MTF of Fig. 2.2 is band-limited to the range $|\kappa| \le D/\lambda$, a pixel spacing of $\Delta \le \frac{1}{2}\frac{\lambda}{D}$ (reciprocal of the Nyquist rate) would be sufficient to avoid aliasing if we were not truncating the point-spread function. However, truncation of the point-spread function means that the actual MTF is no longer band-limited, introducing the possibility of significant distortion of the MTF from aliasing. We thus

pick a pixel spacing of $\Delta = \frac{1}{4}\frac{\lambda}{D}$, which corresponds to a spatial sampling frequency of $4D/\lambda$, which is twice the nyquist rate for the MTF of Fig. 2.2. We next choose $K = 63$, which yields a point-spread function length of 31.75 λ/D, sufficient to reduce aliasing to negligible levels. We will use an object line of 32 λ/D long ($L = 128$), allowing us to study support lengths up to 32 λ/D. Having chosen L and K, the number of pixels in the image line is fixed at $N = 2L + K = 254$, yielding an image line length of 63.5 λ/D.

We are now in a position to compute the energy spectrum of the error vector for various values of SNR, support length, and d/D, using Eq. (2.54) in Eq. (2.11). The results are shown in Fig.'s 2.19 through 2.23. The energy spectrum in each figure is normalized by the total energy of the object vector ($M\sigma_x^2$) to remove the dependence on the variance of an object pixel (see Eq. (2.54)).

The results are rather discouraging. For example, with $d/D = 0.1$ and SNR=10^4 (Fig. 2.23c), the only support length which shows a significant amount of super-resolution is $1\lambda/D$ (roughly the size of the image of a point source through a full aperture of length D). We need to move to an aperture with $d/D = 0.3$ (a not very sparse aperture) to achieve super-resolution with a significant improvement in support length (Fig. 2.21c). Even in the case, a usable support length would be less than $8\lambda/D$. It is clear from these results, that, despite *a priori* statistical knowledge, achieving significant super-resolution performance using only a finite-support constraint on the object requires an enormously large signal-to-noise ratio. As we shall see in the next section, the two-dimensional case is somewhat more encouraging.

2.3.2 Minimum-Variance Results for the Two-Dimensional Case.

We will use the same parameters in the two-dimensional case as we used in the one-dimensional case: object and image pixels $\frac{1}{4}\frac{\lambda}{D}$ on edge ($\Delta = \frac{1}{4}\frac{\lambda}{D}$), a square object plane of 32 λ/D on edge, a square image plane of 63.5 λ/D on edge, and a square point-spread function of 31.75 λ/D on edge. Object support regions will be square with M pixels on edge ($\frac{M}{4}\frac{\lambda}{D}$ on edge). We can now compute the energy spectrum for the two-dimensional model for various values of SNR, support size, and d/D using Eq. (2.54) in Eq. (2.31), where the error vector in Eq. (2.54) in given by Eq. (2.28). Since the energy spectrum of the error vector is now a surface, we have chosen to display the results as a set of grey-scale images in Fig.'s 2.24–2.37. Each figure corresponds to a different aperture sparsity (d/D), support size, and SNR. As in the one-dimensional case, the energy spectrum in each figure is normalized by the total energy in the object ($M^2\sigma_x^2$). The worst case in each figure is a normalized error-vector energy-spectrum of unity, corresponding to black. Normalized energy spectra below 10^{-2} is displayed as white. Each figure is $4D/\lambda$ on edge. The region of most interest to us is the portion of the figures in a circle of diameter $2D/\lambda$, centered at the center of the square. To see how the energy surface relates to the aperture MTF, refer to Fig. 2.24a. This figure corresponds to $d/D = 0.1$, a support size of $7\frac{\lambda}{D} \times 7\frac{\lambda}{D}$, and SNR=$10^2$. If we compare this figure to Fig. 2.11, which shows the MTF corresponding to $d/D = 0.1$, the source of the white diamond-shaped regions in Fig. 2.24a becomes clear. Although there is some super-resolution displayed in the figure, the resulting object estimates would probably not be useful. With an array of this sparsity, and a support region of $7\lambda/D$ on edge, we would probably need an SNR somewhere between 10^4 and 10^5 (Fig.'s 2.24c and 2.24d) to yield useful object estimates. To see how sensitive the results are to aperture sparsity, refer to Fig. 2.28. Here, useful object estimates appear to be achievable with a much-reduced SNR=10^2 if we use an aperture with $d/D = 0.2$ (the MTF of which is shown in Fig. 2.13).

2.4. Least-Squares Processor and Results, One-Dimensional Case, Positivity Constraint.

In computing the results of Section 2.3, we assumed knowledge of the mean and covariance of the object and noise vectors, and we assumed only a finite-support constraint on the object. A probably more realistic situation is one where we are presented with an image and given the MTF of the optical system, and we are required to find the object which best explains the image in some sense, given no statistical knowledge. Furthermore, we may wish to use both finite support *and*

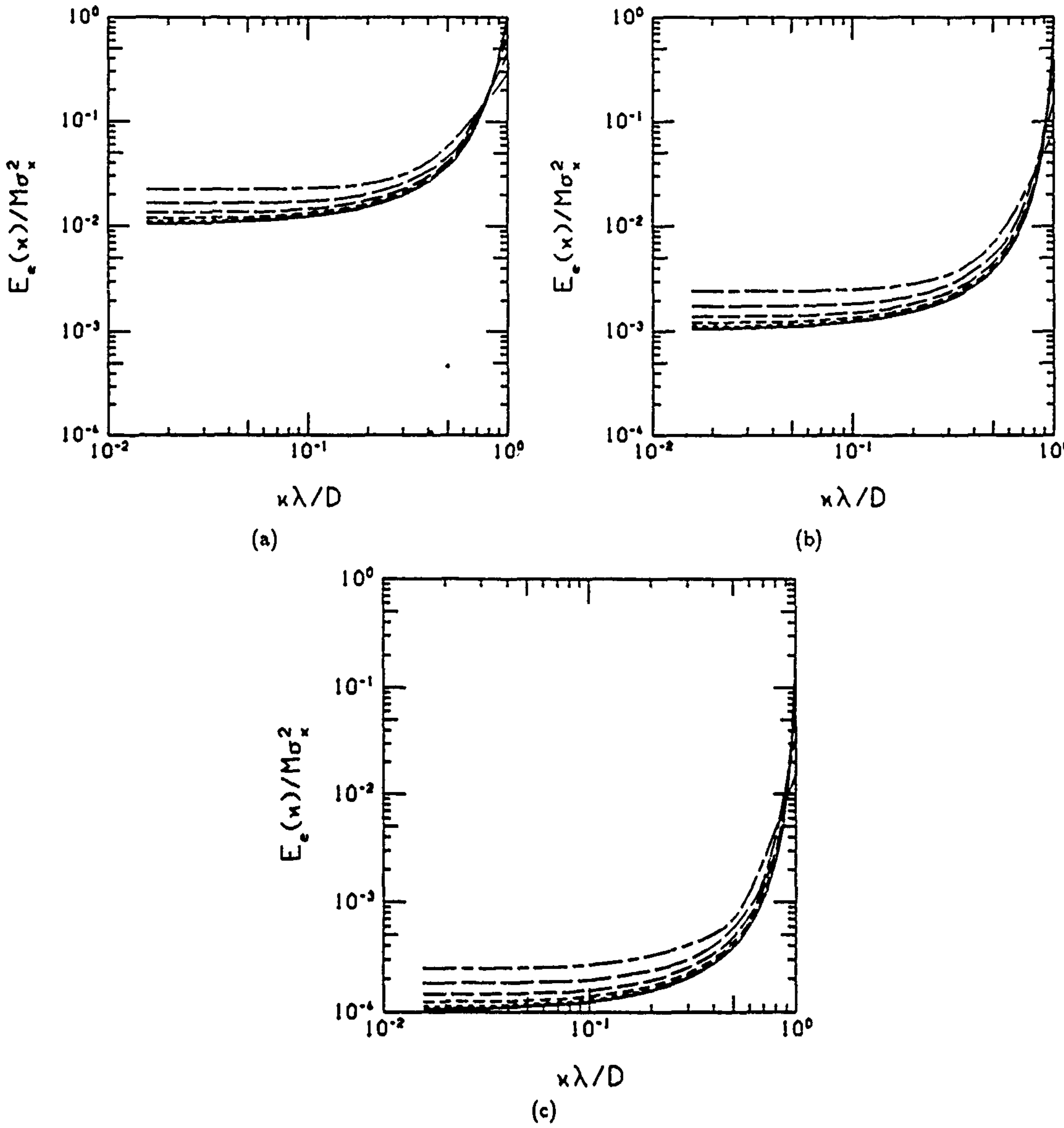


Figure 2.19. Normalized Energy Spectrum of The Error Vector with $d/D = 0.5$ (full aperture). Curve sets (a), (b), and (c) correspond to SNR's of 10^2, 10^3, and 10^4, respectively. The six curves in each curve set correspond to support lengths of (from top to bottom on the left) 1, 2, 4, 8, 16, and 32 λ/D.

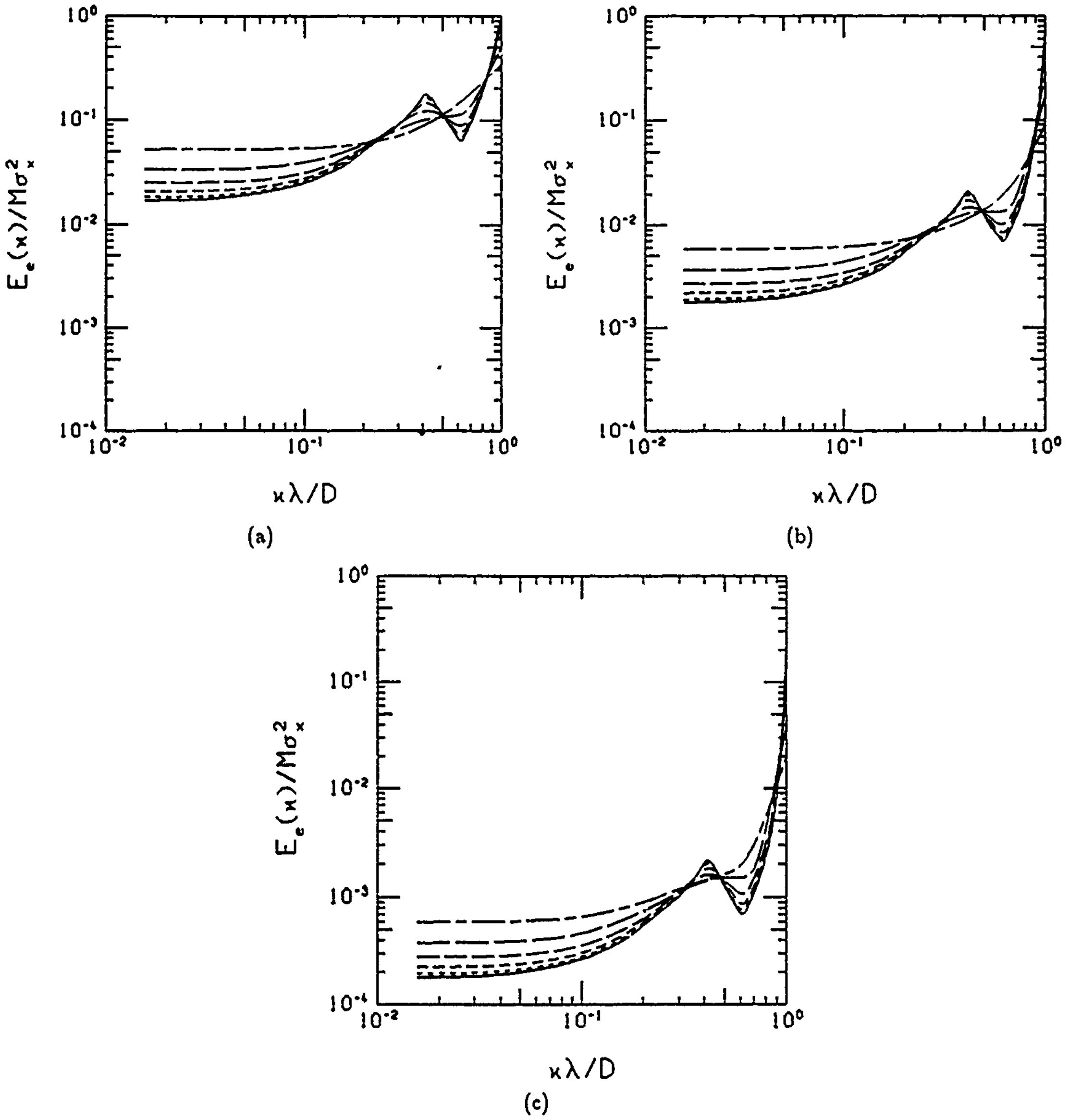


Figure 2.20. Normalized Energy Spectrum of The Error Vector with $d/D = 0.4$. Curve sets (a), (b), and (c) correspond to SNR's of 10^2, 10^3, and 10^4, respectively. The six curves in each curve set correspond to support lengths of (from top to bottom on the left) 1, 2, 4, 8, 16, and 32 λ/D.

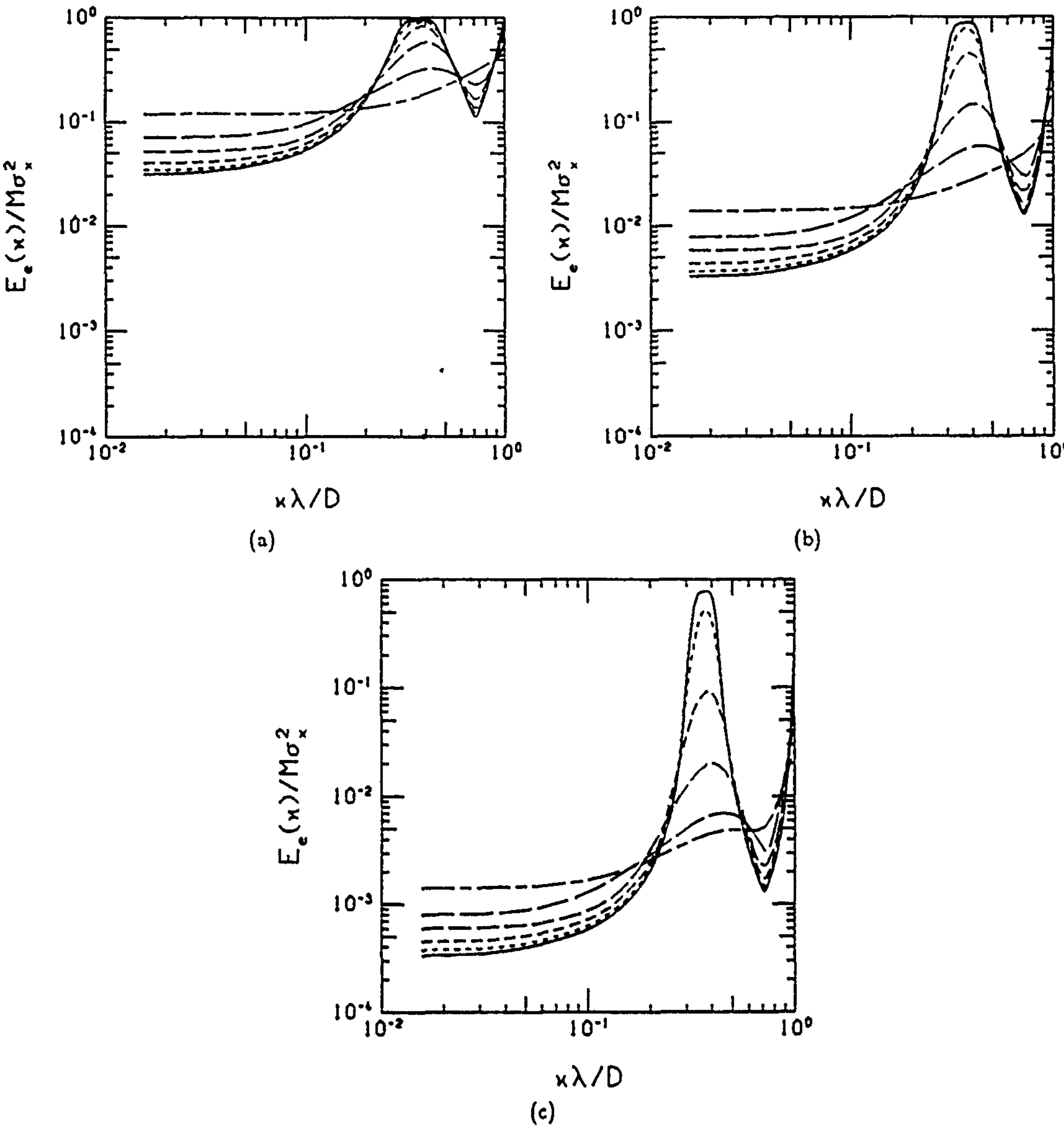


Figure 2.21. Normalized Energy Spectrum of The Error Vector with $d/D = 0.3$. Curve sets (a), (b), and (c) correspond to SNR's of 10^2, 10^3, and 10^4, respectively. The six curves in each curve set correspond to support lengths of (from top to bottom on the left) 1, 2, 4, 8, 16, and 32 λ/D.

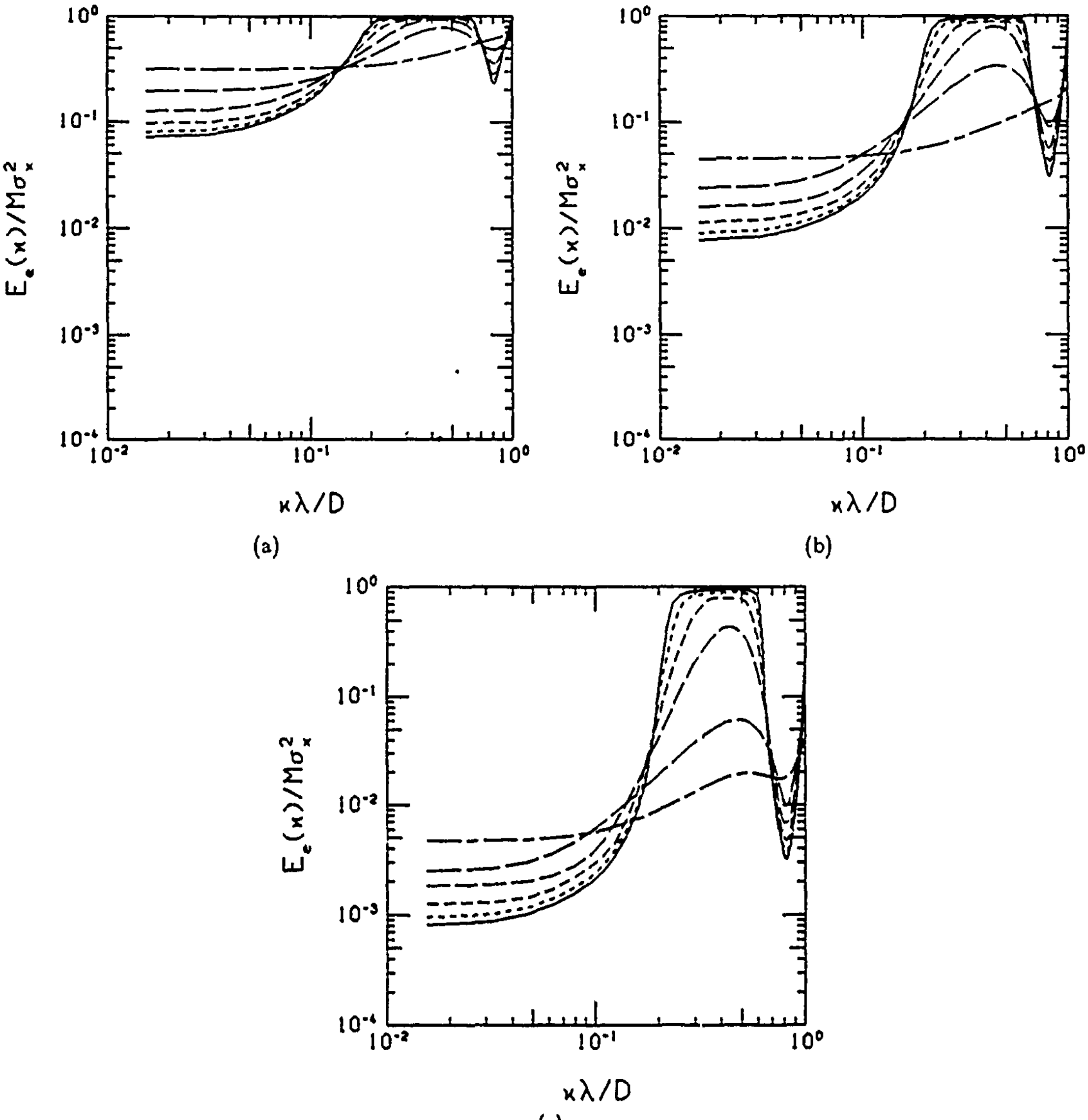


Figure 2.22. Normalized Energy Spectrum of The Error Vector with $d/D = 0.2$.
Curve sets (a), (b), and (c) correspond to SNR's of 10^2, 10^3, and 10^4, respectively. The six curves in each curve set correspond to support lengths of (from top to bottom on the left) 1, 2, 4, 8, 16, and 32 λ/D.

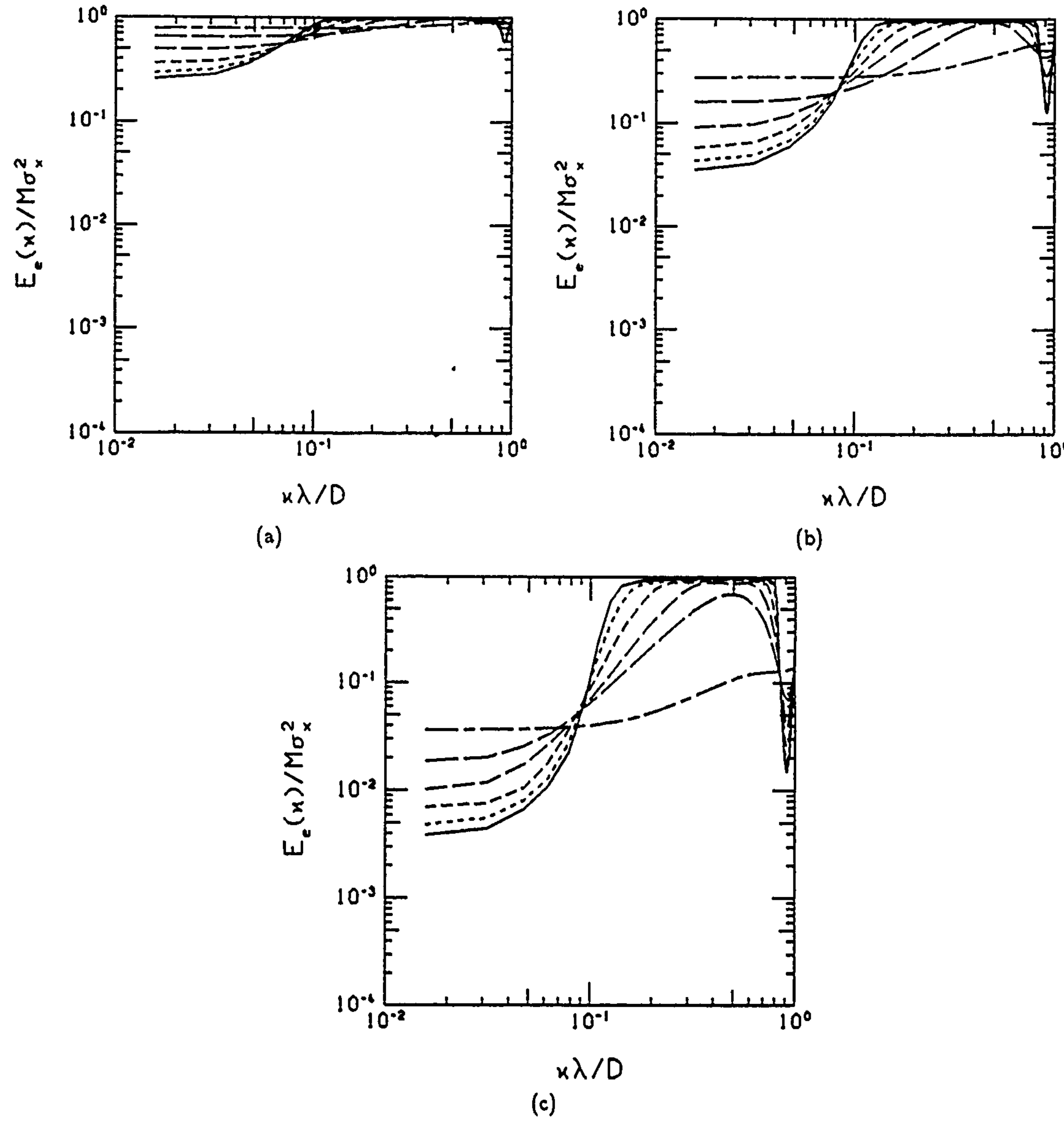


Figure 2.23. Normalized Energy Spectrum of The Error Vector with $d/D = 0.1$.
Curve sets (a), (b), and (c) correspond to SNR's of 10^2, 10^3, and 10^4, respectively. The six curves in each curve set correspond to support lengths of (from top to bottom on the left) 1, 2, 4, 8, 16, and 32 λ/D.

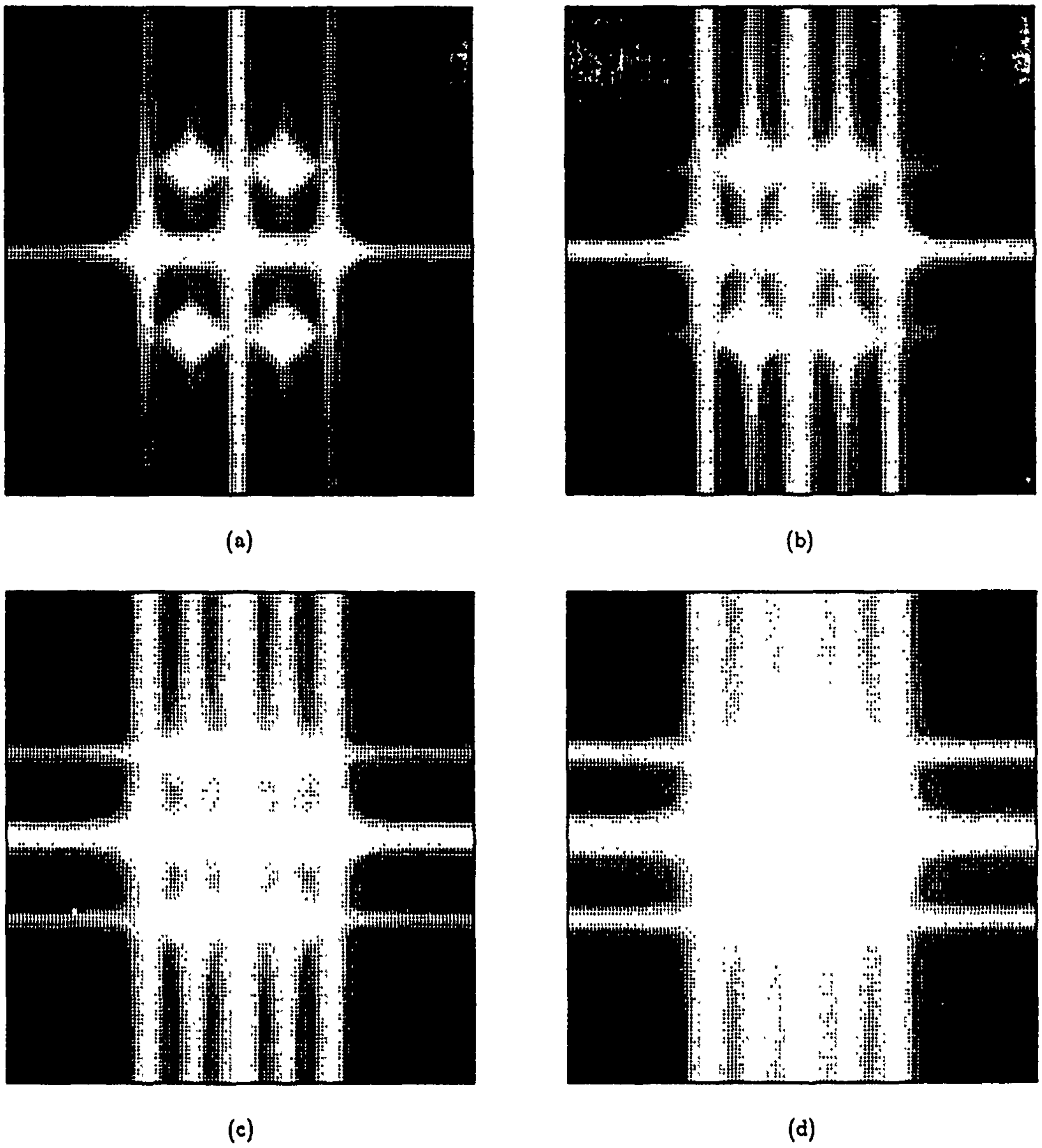
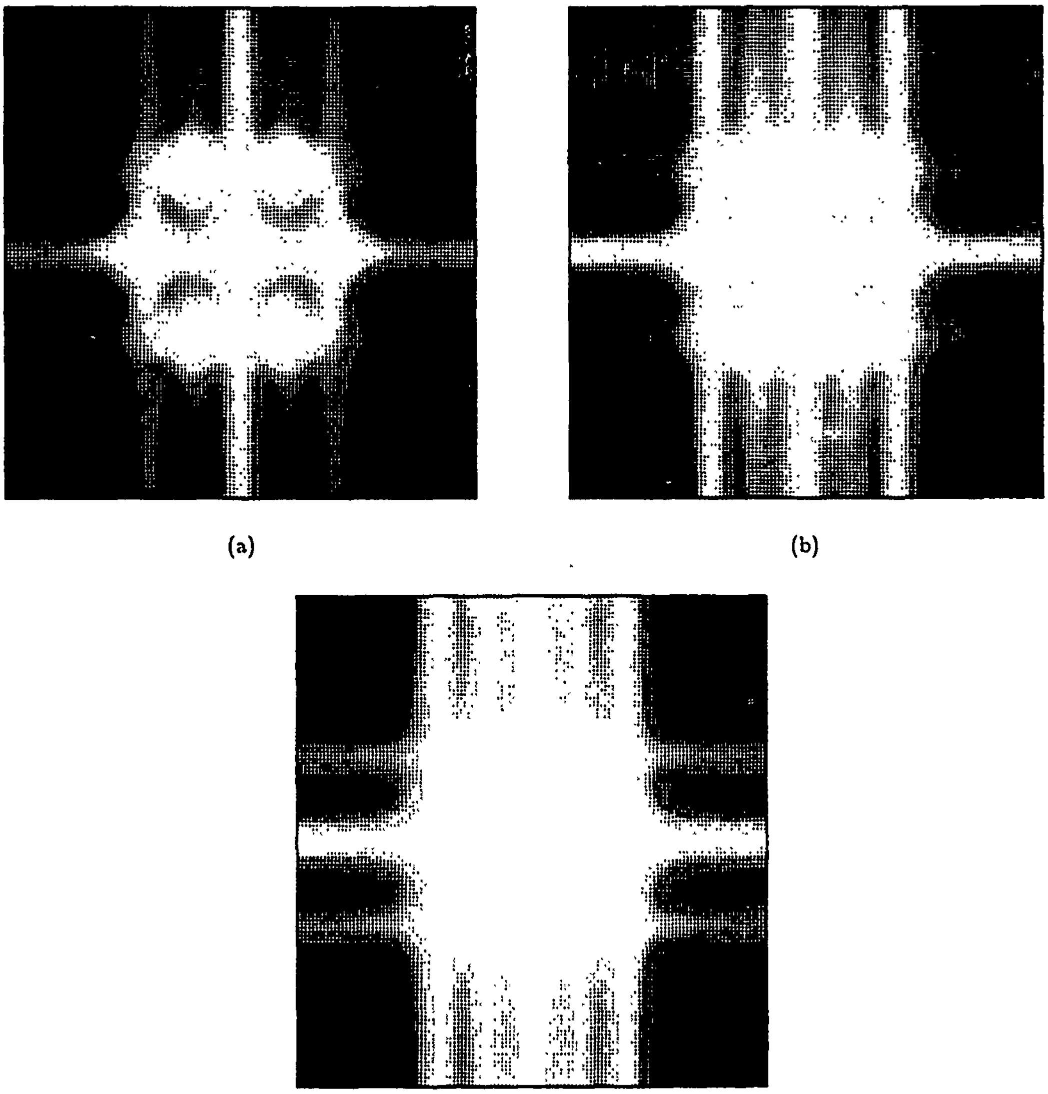
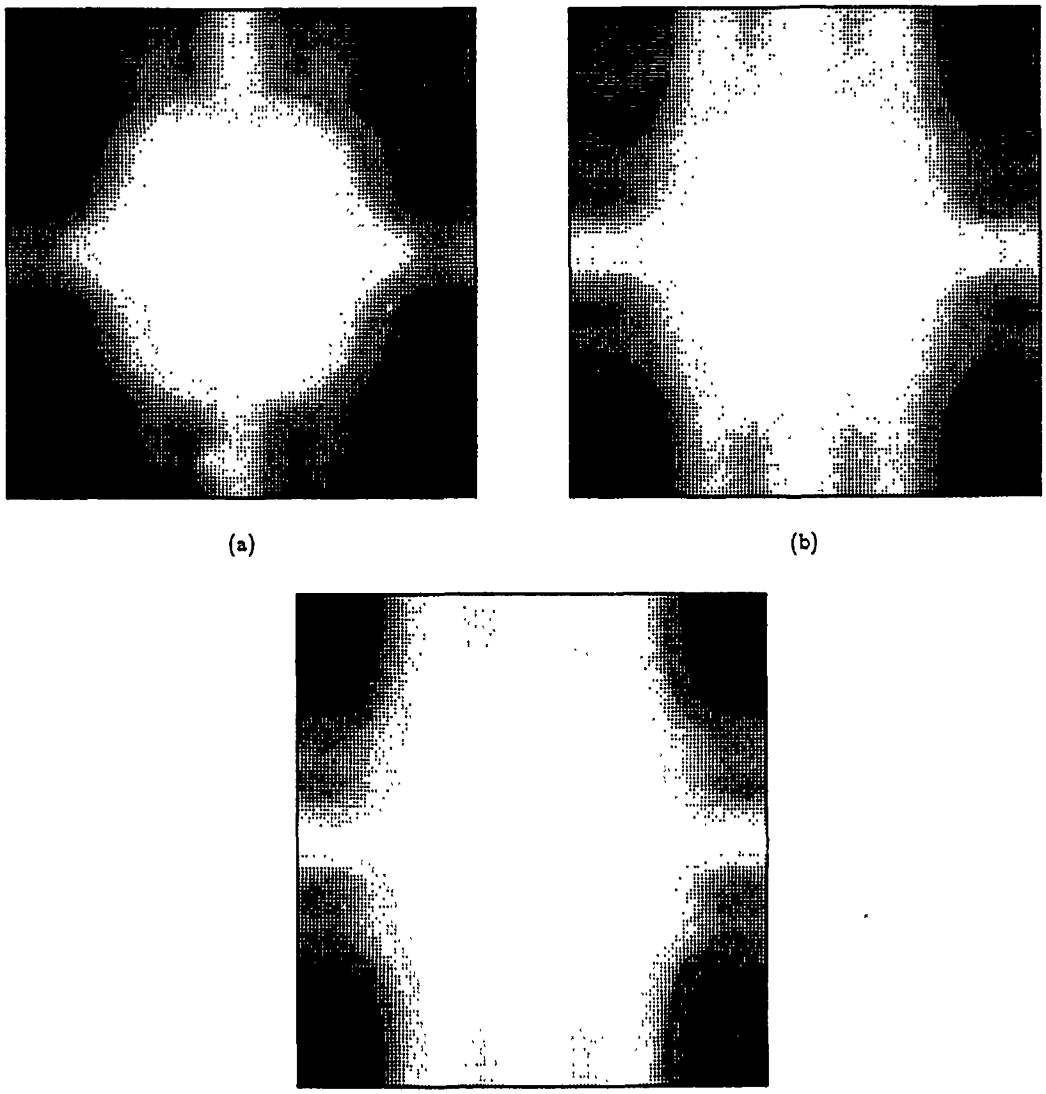


Figure 2.24. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy. Support size is $7\lambda/D \times 7\lambda/D$ and $d/D = 0.1$. Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2}. Fig.'s (a), (b), (c), and (d) correspond to SNR's of 10^2, 10^3, 10^4, and 10^5, respectively.



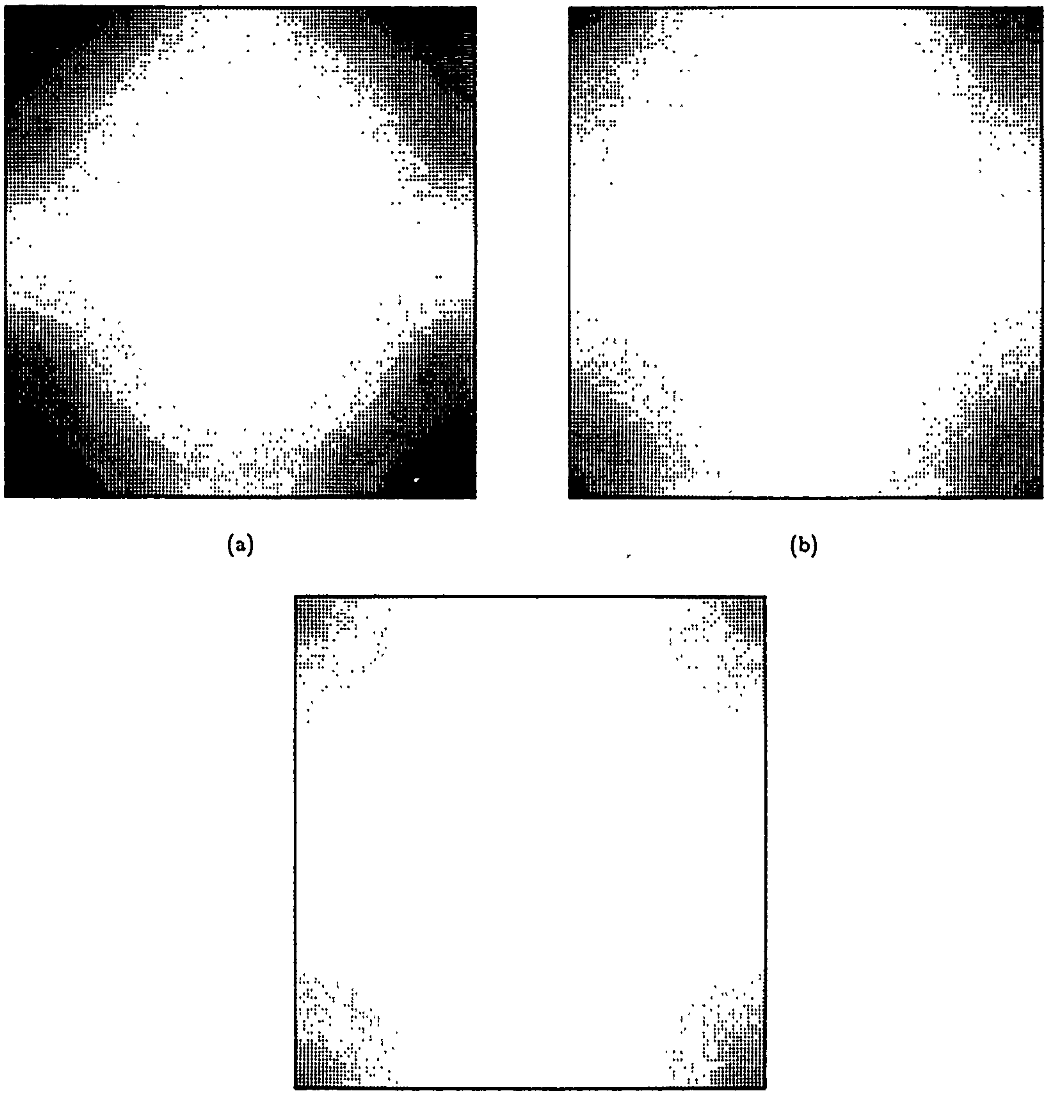
(a) (b) (c)

Figure 2.25. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy. Support size is $4\lambda/D \times 4\lambda/D$ and $d/D = 0.1$. Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2}. Fig.'s (a), (b), and (c) correspond to SNR's of 10^2, 10^3, and 10^4, respectively.



(c)

Figure 2.26. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy. Support size is $2\lambda/D \times 2\lambda/D$ and $d/D = 0.1$. Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2}. Fig.'s (a), (b), and (c) correspond to SNR's of 10^2, 10^3, and 10^4, respectively.



(a) (b) (c)

Figure 2.27. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy. Support size is $1\lambda/D \times 1\lambda/D$ and $d/D = 0.1$. Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2}. Fig.'s (a), (b), and (c) correspond to SNR's of 10^2, 10^3, and 10^4, respectively.

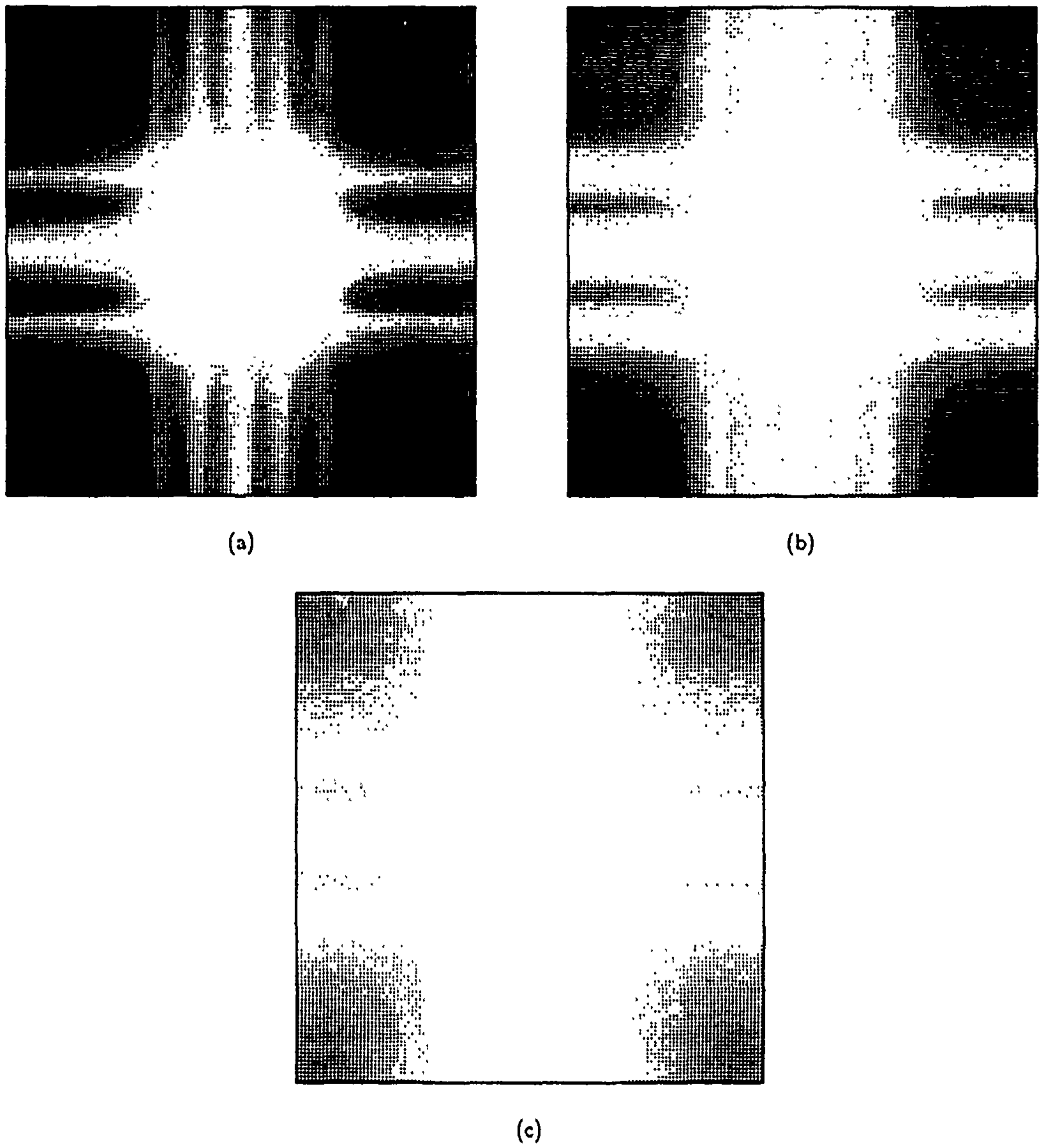


Figure 2.28. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy. Support size is $7\lambda/D \times 7\lambda/D$ and $d/D = 0.2$. Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2}. Fig.'s (a), (b), and (c) correspond to SNR's of 10^2, 10^3, and 10^4, respectively.

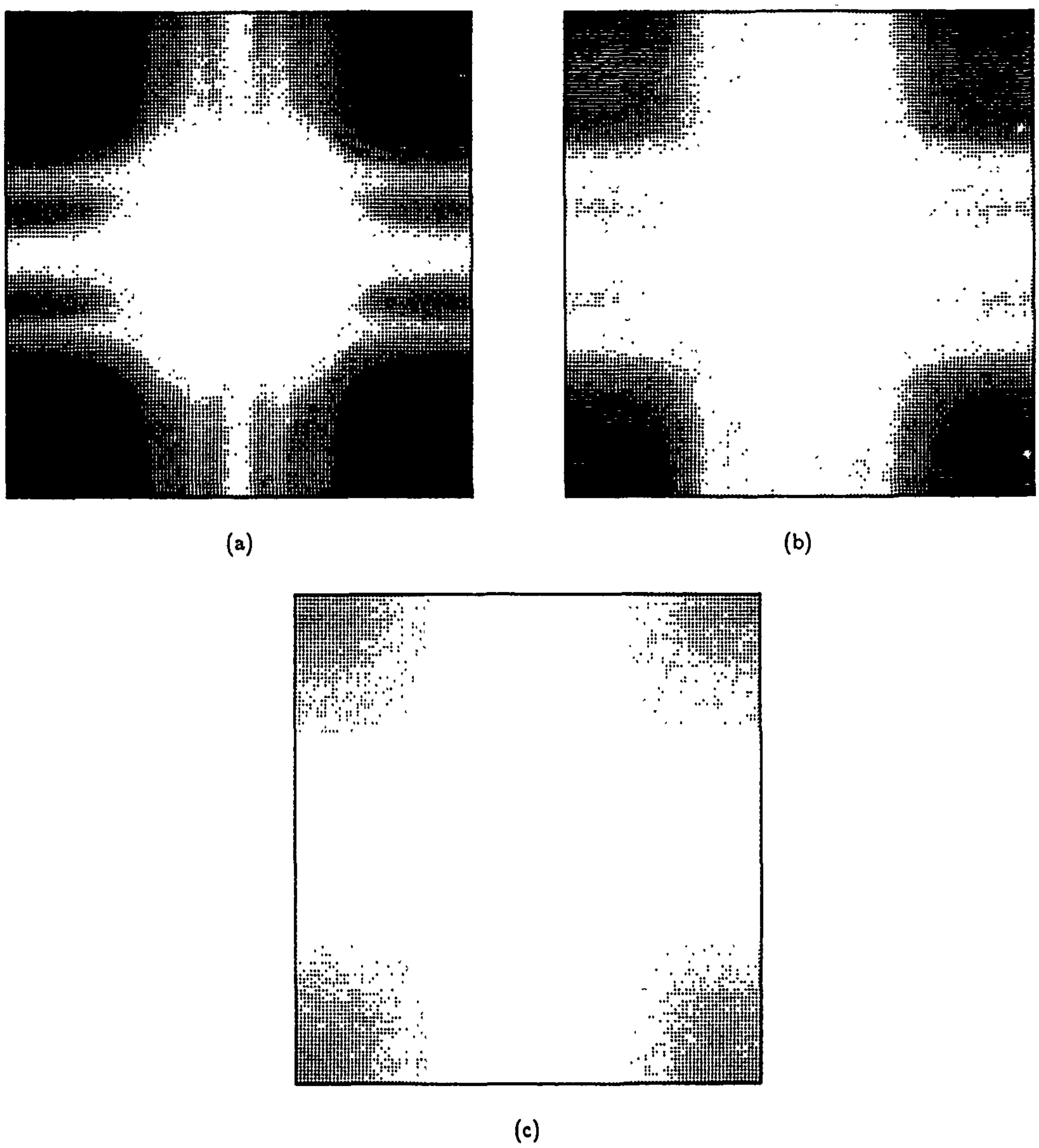
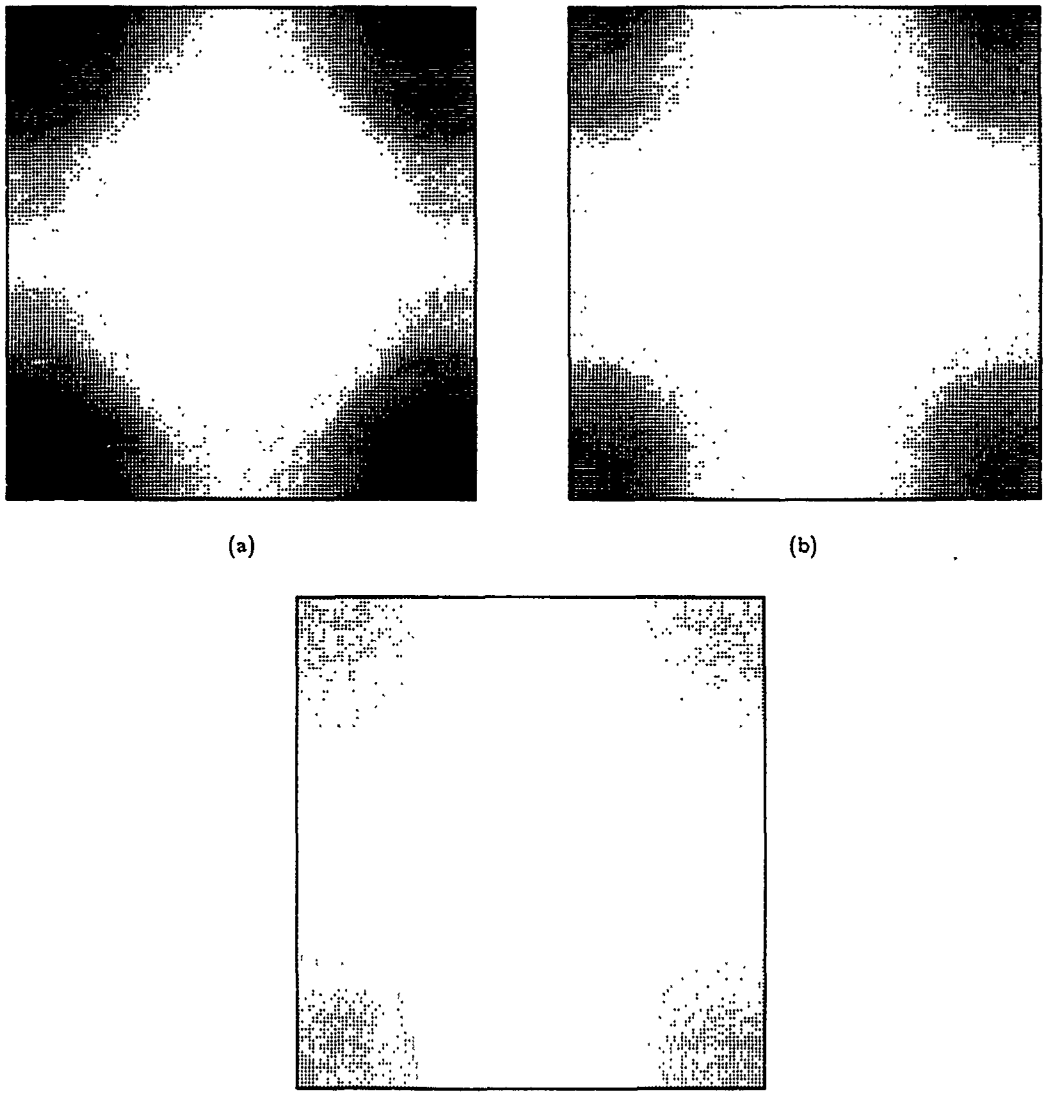


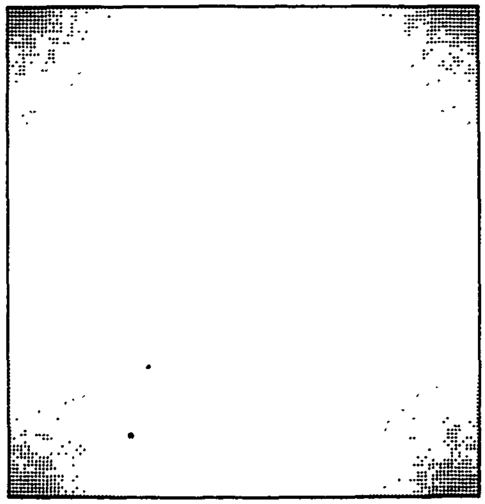
Figure 2.29. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy. Support size is $4\lambda/D \times 4\lambda/D$ and $d/D = 0.2$. Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2}. Fig.'s (a), (b), and (c) correspond to SNR's of 10^2, 10^3, and 10^4, respectively.



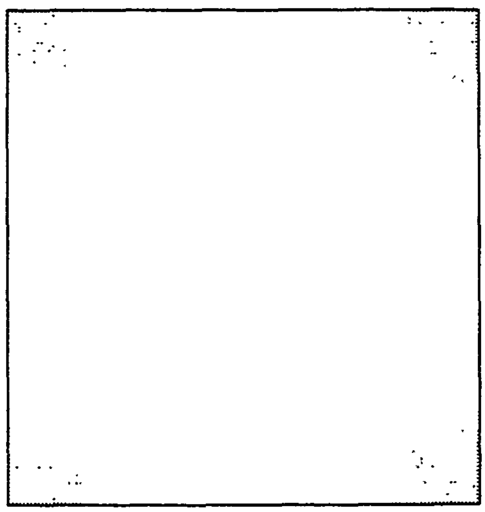
(a) (b)

(c)

Figure 2.30. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy. Support size is $2\lambda/D \times 2\lambda/D$ and $d/D = 0.2$. Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2}. Fig.'s (a), (b), and (c) correspond to SNR's of 10^2, 10^3, and 10^4, respectively.



(a)



(b)

Figure 2.31. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy. Support size is $1\lambda/D \times 1\lambda/D$ and $d/D = 0.2$. Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2}. Fig.'s (a) and (b) correspond to SNR's of 10^2 and 10^3, respectively.

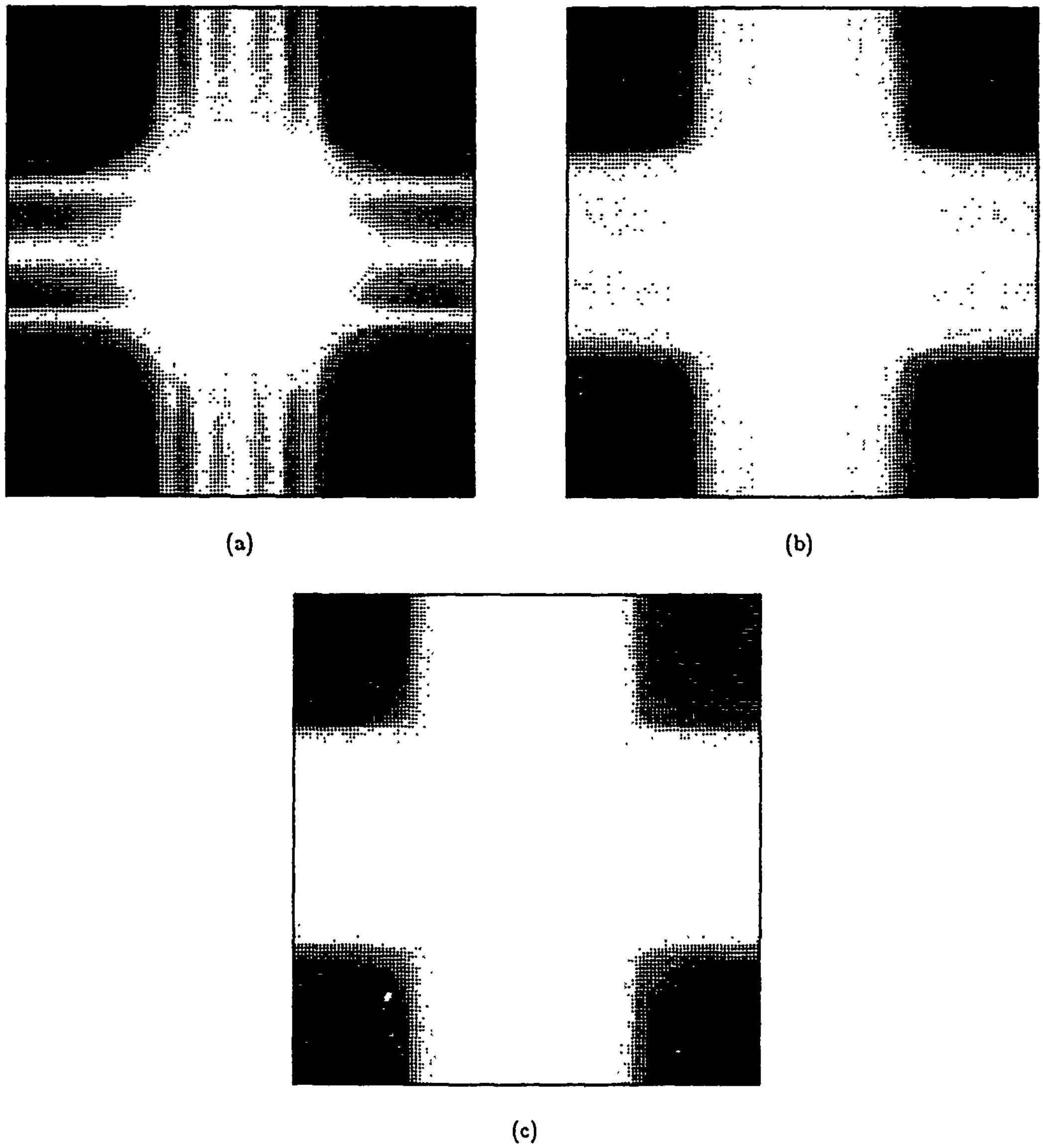
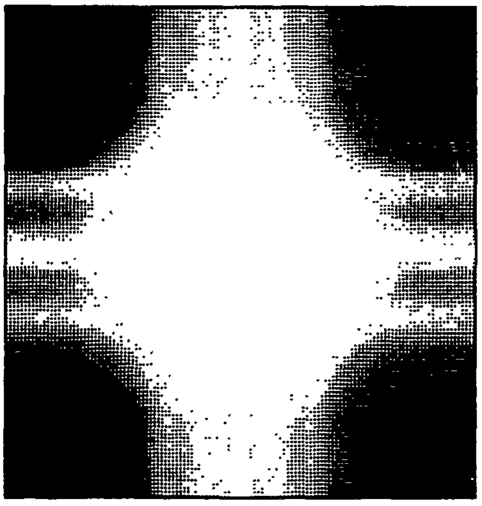
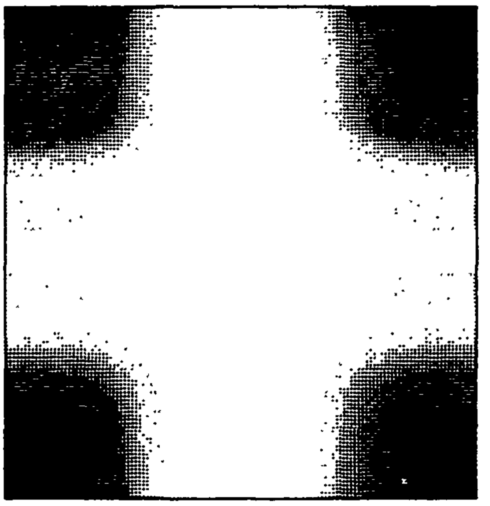


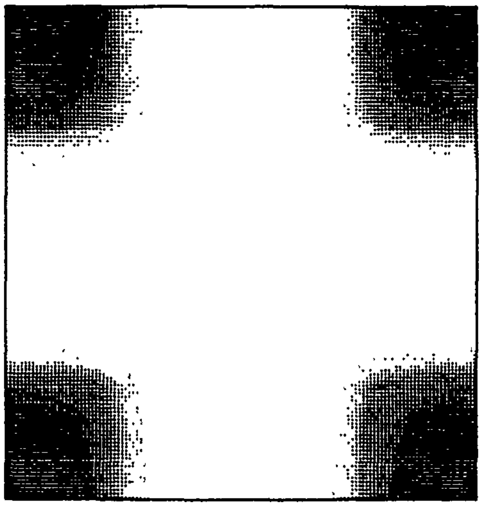
Figure 2.32. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy. Support size is $7\lambda/D \times 7\lambda/D$ and $d/D = 0.3$. Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2}. Fig.'s (a), (b), and (c) correspond to SNR's of 10^2, 10^3, and 10^4, respectively.



(a)

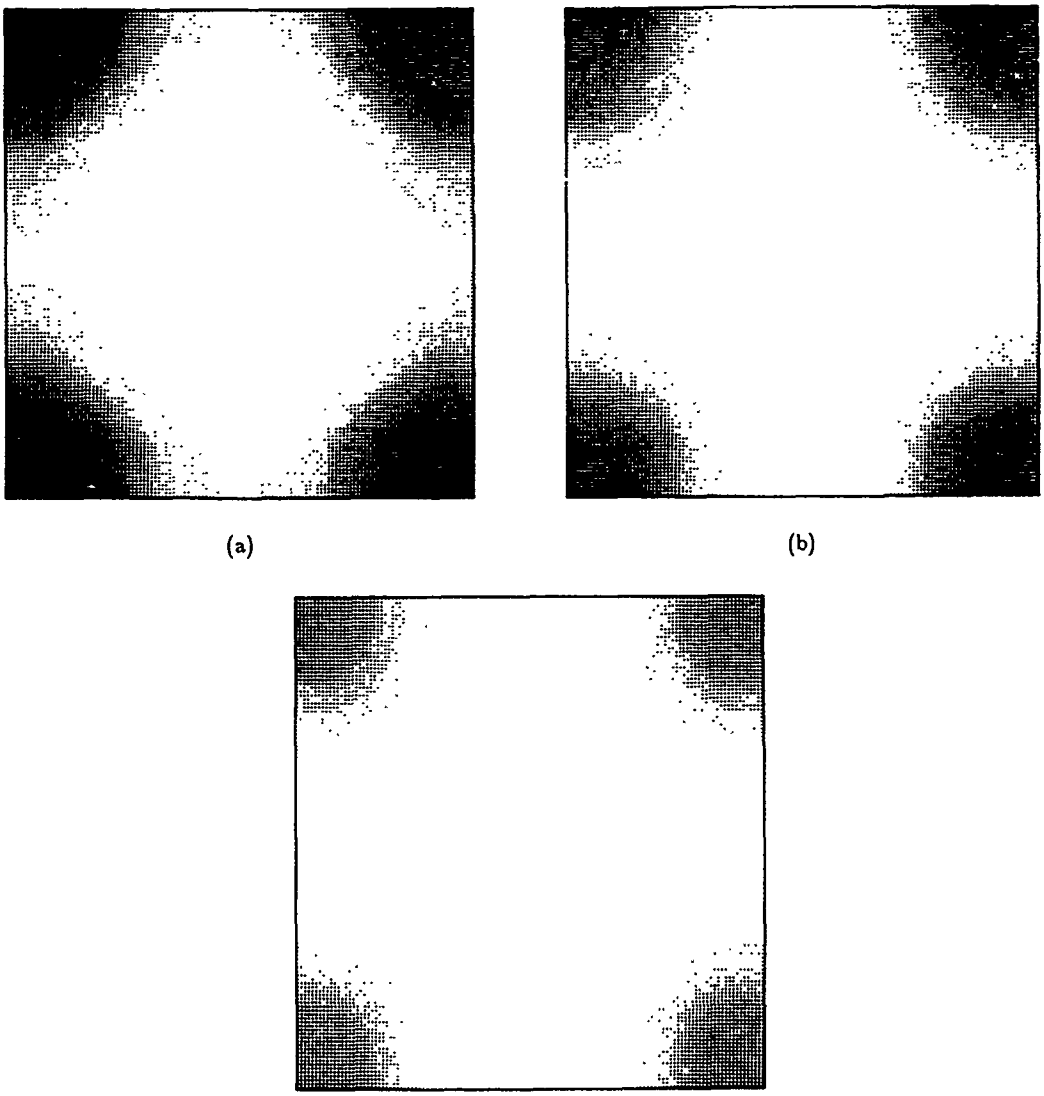


(b)



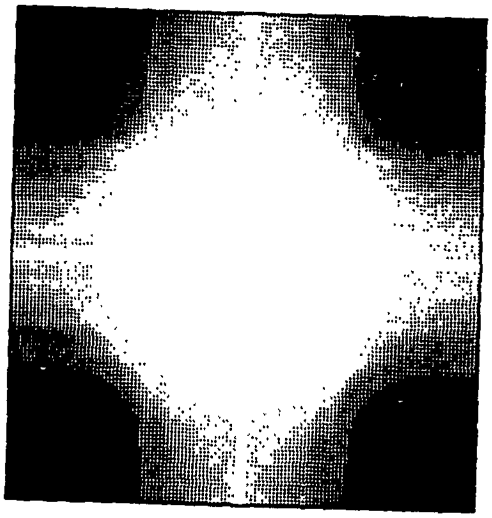
(c)

Figure 2.33. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy. Support size is $4\lambda/D \times 4\lambda/D$ and $d/D = 0.3$. Each image is $4D/\lambda$ cn edge. Black denotes unity and white denotes less than 10^{-2}. Fig.'s (a), (b), and (c) correspond to SNR's of 10^2, 10^3, and 10^4, respectively.

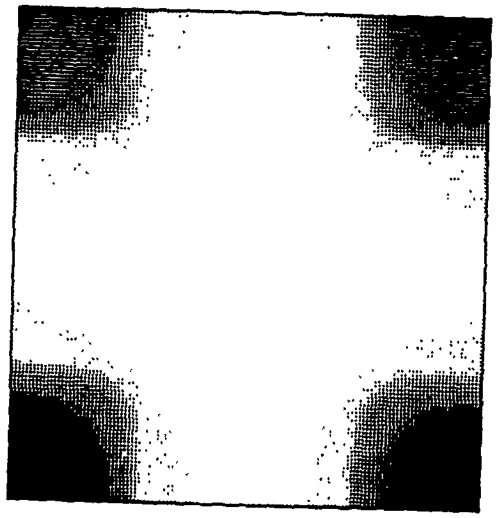


(a) (b) (c)

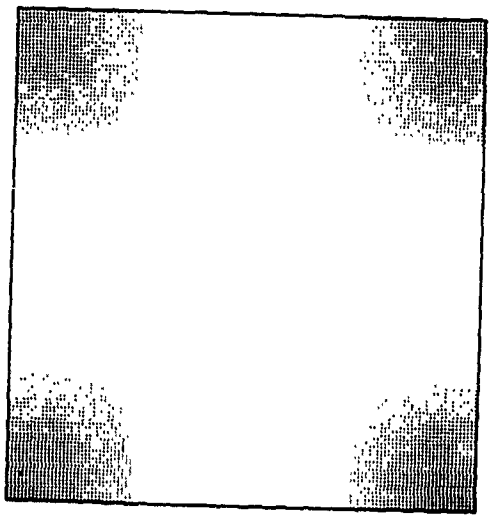
Figure 2.34. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy. Support size is $2\lambda/D \times 2\lambda/D$ and $d/D = 0.3$. Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2}. Fig.'s (a), (b), and (c) correspond to SNR's of 10^2, 10^3, and 10^4, respectively.



(a)

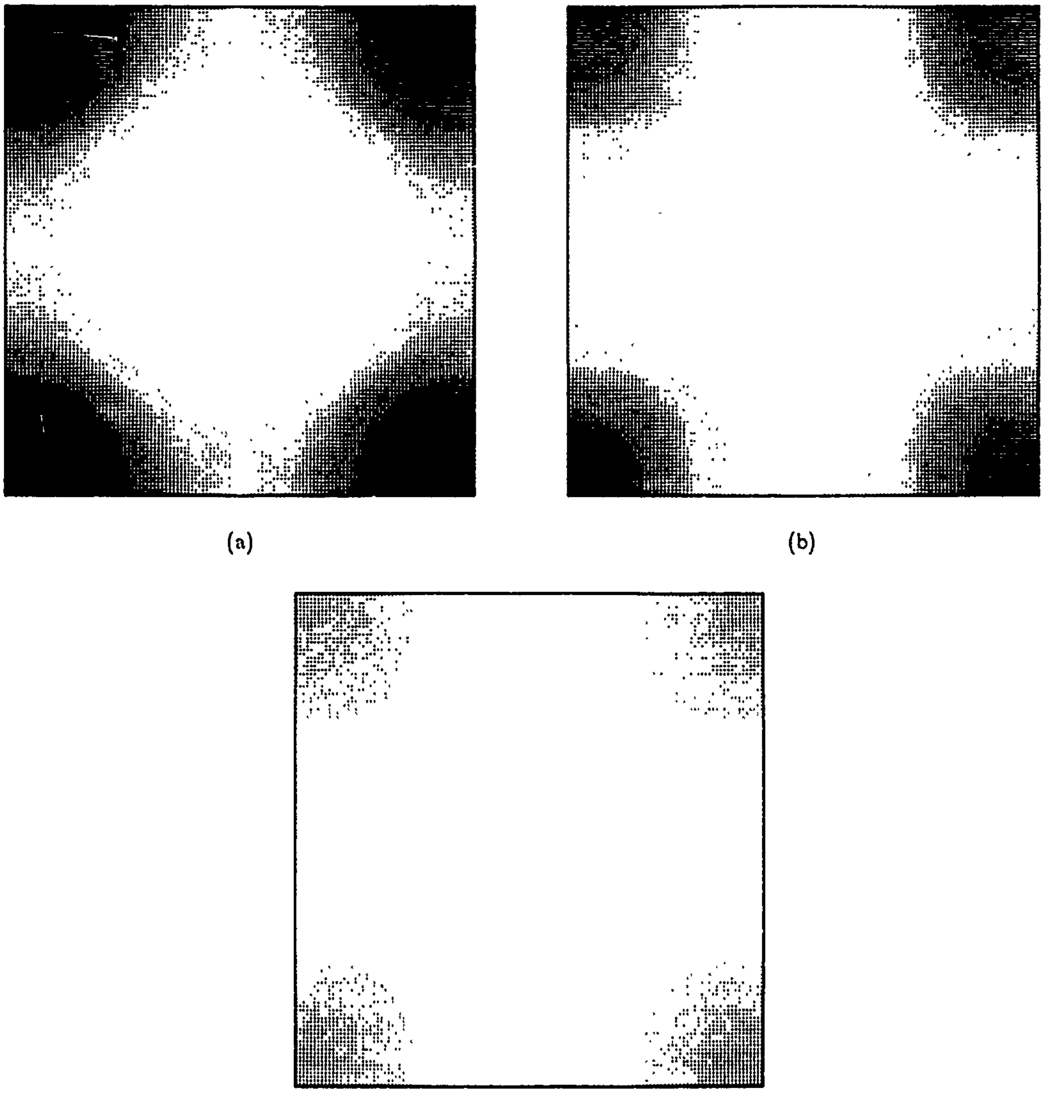


(b)



(c)

Figure 2.35. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy. Support size is $7\lambda/D \times 7\lambda/D$ and $d/D = 1.0$ (full aperture). Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2}. *Fig.'s* (a), (b), and (c) correspond to SNR's of 10^2, 10^3, and 10^4, respectively.

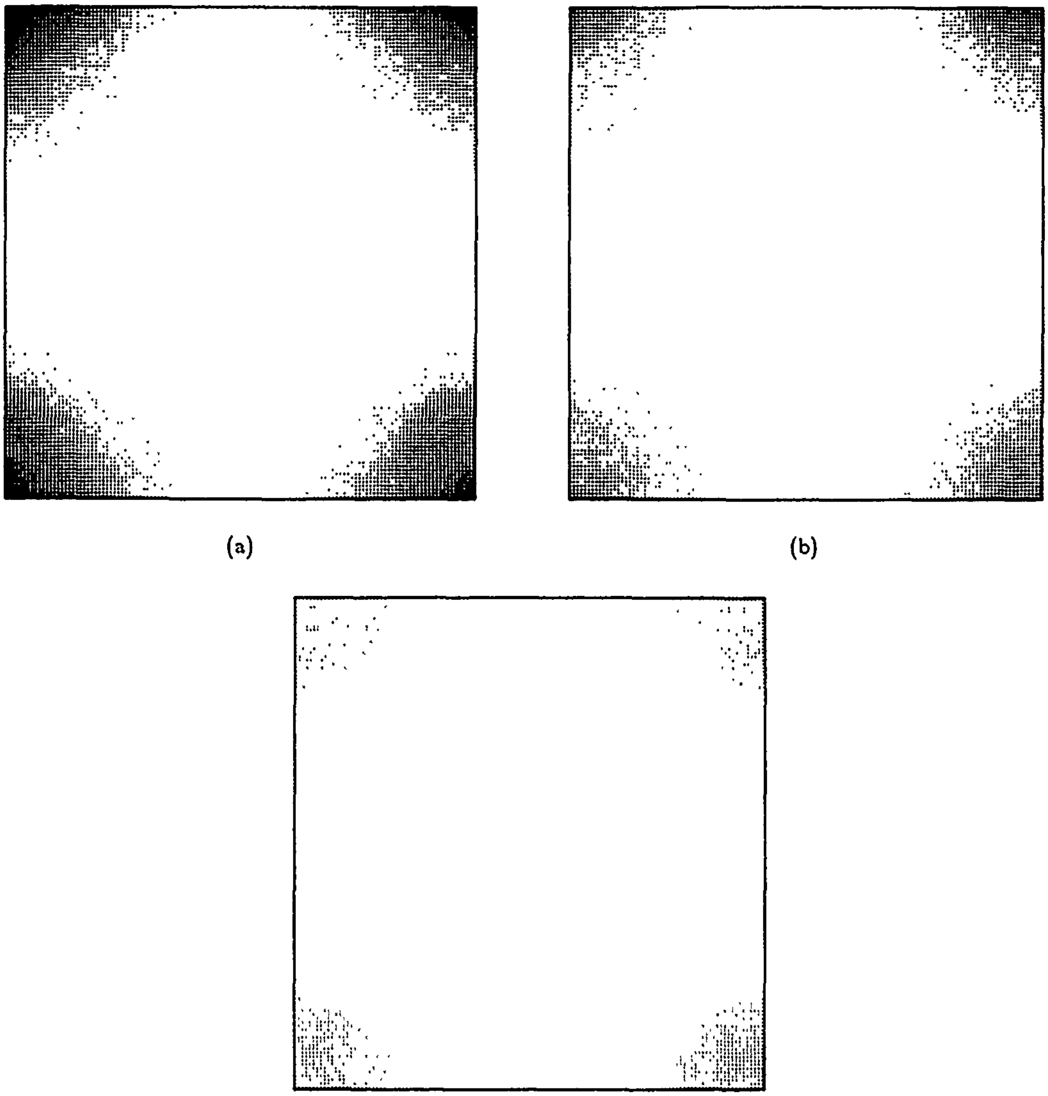


(a)

(b)

(c)

Figure 2.36. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy. Support size is $4\lambda/D \times 4\lambda/D$ and $d/D = 1.0$ (full aperture). Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2}. Fig.'s (a), (b), and (c) correspond to SNR's of 10^2, 10^3, and 10^4, respectively.



(a) (b)

(c)

Figure 2.37. Grey-Scale Images of the Error Vector Energy Spectrum Normalized by Total Object Energy. Support size is $2\lambda/D \times 2\lambda/D$ and $d/D = 1.0(fullaperture)$. Each image is $4D/\lambda$ on edge. Black denotes unity and white denotes less than 10^{-2}. Fig.'s (a), (b), and (c) correspond to SNR's of 10^2, 10^3, and 10^4, respectively.

positivity as constraints on the allowable objects. In this section we will use a least-squares criterion for deciding which object "best" explains the image, i.e., we pick the object vector estimate $\hat{\mathbf{x}}$ which minimizes

$$\epsilon = \|\mathbf{y} - G\hat{\mathbf{x}}\|^2. \tag{2.55}$$

Here we use the same one-dimensional optical model as that described in Section 2.2, with $\mathbf{y}$ the vector of image pixels, $\mathbf{x}$ the vector of object pixels, and $G = BW$ a matrix containing the system point-spread function and the object support information. We will distinguish between two object estimators: $\hat{\mathbf{x}}_s$, which minimizes Eq. (2.55) using only finite support as a constraint on the object, and $\hat{\mathbf{x}}_p$, which minimizes Eq. (2.55) when the object is constrained by both finite support and positivity. We can obtain $\hat{\mathbf{x}}_s$ in closed form. Carrying out the operation indicated in Eq. (2.55) yields

$$\begin{aligned}\epsilon &= (\mathbf{y} - G\hat{\mathbf{x}})^T(\mathbf{y} - G\hat{\mathbf{x}}) \\ &= \mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T G\hat{\mathbf{x}} + \hat{\mathbf{x}}^T G^T G\hat{\mathbf{x}}.\end{aligned} \tag{2.56}$$

The gradient of ϵ with respect to the components of $\hat{\mathbf{x}}$ is

$$\nabla\epsilon = -2G^T\mathbf{y} + 2G^T G\hat{\mathbf{x}}, \tag{2.57}$$

where here the gradient is taken to be a column vector. Setting Eq. (2.57) to zero and solving for $\hat{\mathbf{x}}$ yields

$$\hat{\mathbf{x}}_s = (G^T G)^{-1} G^T\mathbf{y}. \tag{2.58}$$

Since G has full column rank [see Eq. (2.14)], $G^T G$ is nonsingular and its inverse is well defined. The error vector with this estimate is

$$\begin{aligned}\mathbf{e}_s &= \mathbf{x} - \hat{\mathbf{x}}_s \\ &= \mathbf{x} - (G^T G)^{-1} G^T\mathbf{y}.\end{aligned} \tag{2.59}$$

Using $\mathbf{y}$ from Eq. (2.5) in Eq. (2.59), with $G = BW$, yields

$$\begin{aligned}\mathbf{e}_s &= \mathbf{x} - (G^T G)^{-1} G^T(G\mathbf{x} + \mathbf{n}) \\ &= (G^T G)^{-1} G^T\mathbf{n}.\end{aligned} \tag{2.60}$$

The covariance matrix of the error vector is

$$\begin{aligned}\langle \mathbf{e}_s\mathbf{e}_s^T\rangle &= (G^T G)^{-1} G^T\langle\mathbf{n}\mathbf{n}^T\rangle G(G^T G)^{-1} \\ &= (G^T G)^{-1} G^T R_{nn} G(G^T G)^{-1}.\end{aligned} \tag{2.61}$$

If we again assume that the noise vector is zero-mean and white, i.e.

$$R_{nn} = \sigma_n^2 I, \tag{2.62}$$

then

$$\langle \mathbf{e}_s\mathbf{e}_s^T\rangle = \sigma_n^2(G^T G)^{-1}. \tag{2.63}$$

Unfortunately, one cannot obtain $\hat{\mathbf{x}}_p$ in closed form; one must use some numerical iteration method for each case of an object vector and a noise vector. Therefore, to evaluate the error vector covariance matrix when using positivity as a constraint requires a simulation. To generate object vectors, we generated M independent Rayleigh random variables, one for each pixel. Thus, object intensity pixels are always positive. Noise vector were generated using independent, zero-mean, Gaussian random variable for each pixel. An image vector $\mathbf{y}$ was generated using Eq. (2.5) for each

sample object and noise vector generated. The object vector estimate, $\hat{\mathbf{x}}_p$, for each case, was found using a modified gradient-search method where the components in the object vector were not allowed to go negative. If we let $\mathbf{x}(n)$ be the object vector and $\hat{\mathbf{x}}_p(n)$ be the object vector estimate for the n^{th} trial, then the error covariance matrix was estimated using

$$\widehat{\langle \mathbf{e}_p \mathbf{e}_p^T \rangle} = \frac{1}{N} \sum_{n=1}^{N} \left(\mathbf{x}(n) - \hat{\mathbf{x}}_p(n) \right) \left(\mathbf{x}(n) - \hat{\mathbf{x}}_p(n) \right)^T . \tag{2.64}$$

Using Eq. (2.63) and (2.64) in Eq. (2.11), we can compute the energy spectrum of the error vector with and without the positivity constraint, and compare results.

Unfortunately, at the time of this report, our results are rather meager due to difficulties with the modified gradient search method. At this time we have results only for an object support length of four pixels ($1\lambda/D$). The results are shown in Fig.'s 2.38–2.40. Again, we have normalized the error-vector energy spectra with the total energy in the object vector (sum of the mean-square intensity values of each pixel). Each figure contains a set of curves corresponding to one d/D ratio (0.1 for Fig. 2.38, 0.2 for Fig. 2.39, and 0.3 for Fig. 2.40). Within each figure are three sets of two curves each, each set corresponding to a different SNR (from bottom to top, 10^4, 10^3, and 10^2, respectively). Here, SNR is defined as the ratio of the second moment of an object intensity pixel to the second moment of a noise pixel. The solid curve in each curve-pair corresponds to the finite-support constraint only, and the dashed curve is the result when the positivity constraint is added. With these very limited results, it would appear that the use of a positivity constraint in addition to the finite-support constraint may result in a significant improvement in performance. However, we note that the amount of improvement seems to decrease as SNR increases.

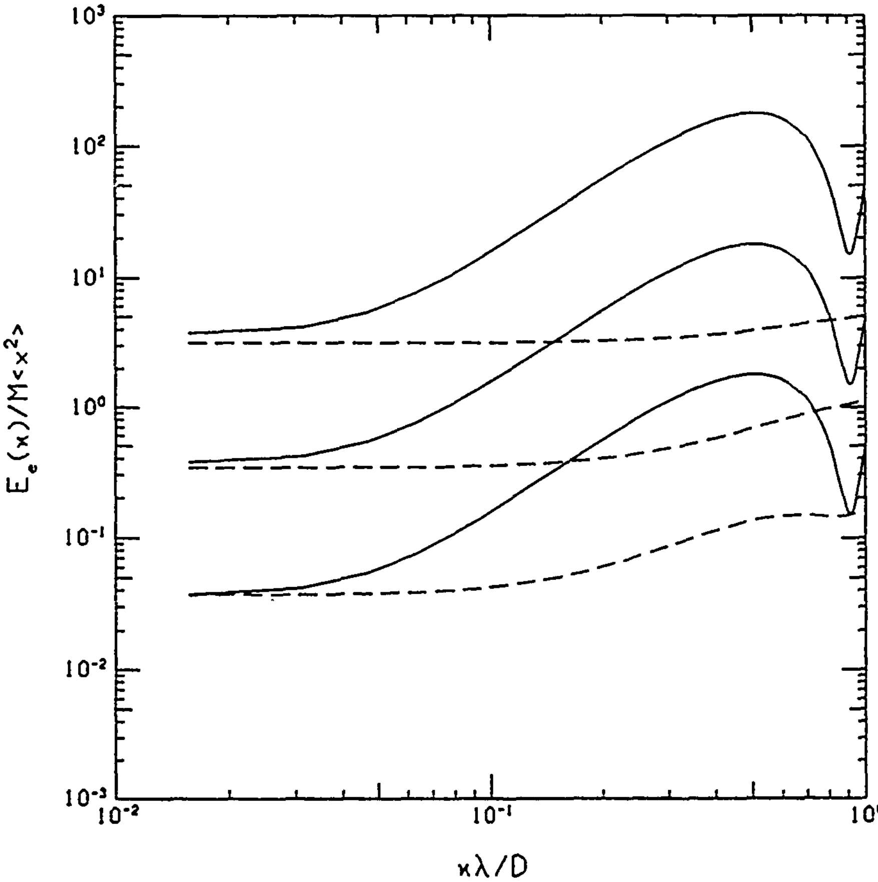


Figure 2.38. Normalized Error Vector Energy Spectrum for $d/D = 0.1$ and a Support Length of $1\lambda/D$. The three sets of curves correspond to SNR's of (from top to bottom) 10^2, 10^3, and 10^4. The solid and dashed curves are without and with positivity constraint, respectively.

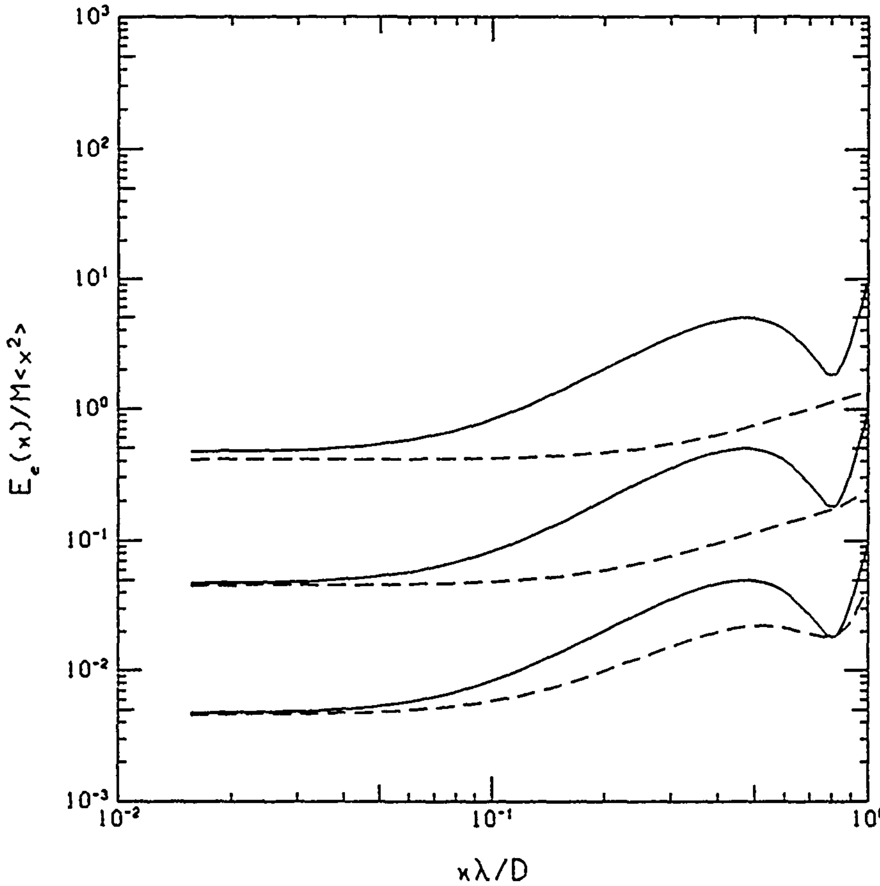


Figure 2.39. Normalized Error Vector Energy Spectrum for $d/D = 0.2$ and a Support Length of $1\lambda/D$. The three sets of curves correspond to SNR's of (from top to bottom) 10^2, 10^3, and 10^4. The solid and dashed curves are without and with positivity constraint, respectively.

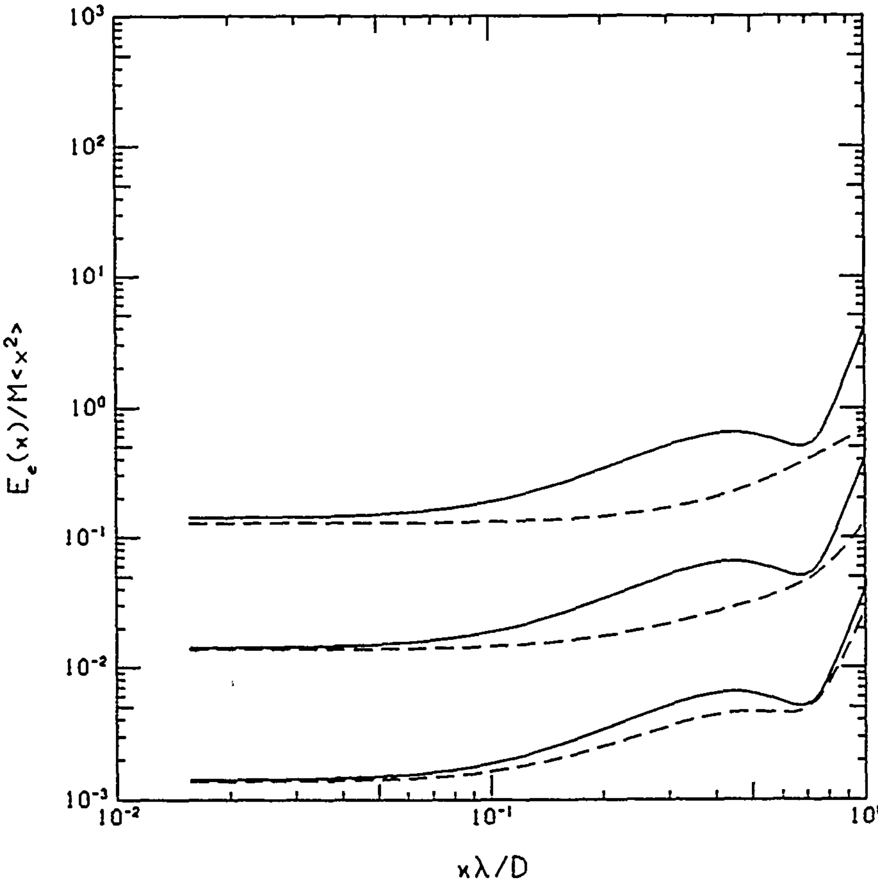


Figure 2.40. Normalized Error Vector Energy Spectrum for $d/D = 0.3$ and a Support Length of $1\lambda/D$. The three sets of curves correspond to SNR's of (from top to bottom) 10^2, 10^3, and 10^4. The solid and dashed curves are without and with positivity constraint, respectively.

Chapter 3

Super-Resolution with Sparse Arrays Revisited—Conclusion of Two Aperture Case

(originally issued as TR-1052)

3.1. Introduction

In Chapter 2 we reported on some results of an investigation into the feasibility of obtaining useful reconstruction of objects viewed through a sparse array of apertures. A requirement placed on the reconstruction was that the highest useful spatial frequency in the reconstruction be commensurate with the outer diameter of the sparse array. All of the one -demensional results contained in Chapter 2 were restricted to the case of two fixed subapertures, a case that we now know to be of little interest. In this chapter we present some miscellaneous two-aperture results that were obtained subsequent to the work of Chapter 2. We report these results, without comment, for purposes of historical completeness and the remote chance that they may be of use.

All of the results in this chapter are for the one-dimensional model described in Section 2.2.1 of Chapter 2. Section 3.2 of this chapter describes some changes made to the model subsequent to publication of Chapter 2. Sections 3 and 4 of this chapter give results for the minimum variance and least squares methods, respectively. Appendix B contains listings of fortran programs that, in addition to those given in Chapter 2, were used to generate the data of this chapter.

3.2. Some Changes to the One-dimensional Model

All of the one-dimensional results of Chapter 2 were generated using the point spread function (PSF) given by Eq.'s (2.7) and (2.8) of Chapter 2. For purposes of this chapter, we replace that earlier PSF with two point spread functions, called $h_1,(n)$ and $h_2(n)$.

$$h_1(n) = \Delta w_\ell(x) h_c(x)|_{x=n\Delta}, \tag{3.1}$$

where Δ is the sample interval,

$$h_c(x) = \frac{4d^2}{D\lambda}\cos^2\left[\frac{\pi(D-d)}{\lambda}x\right]\left[\frac{\sin(\pi x d/\lambda)}{\pi x d/\lambda}\right]^2, \tag{3.2}$$

and $w_\ell(x)$ is the ℓ^{th} of five window functions given below.

<u>Window One (Rectangular)</u>

$$w_1(x) = \begin{cases} 1, & |x| \le k\Delta \\ 0, & \text{else.} \end{cases} \tag{3.3}$$

<u>Window Two (Bartlett)</u>

$$w_2(x) = \begin{cases} 1 - |x|/k\Delta, & |x| \le k\Delta \\ 0, & \text{else.} \end{cases} \tag{3.4}$$

<u>Window Three (Hanning)</u>

$$w_3(x) = \begin{cases} \frac{1}{2} + \frac{1}{2}\cos(\pi x/k\Delta), & |x| \le k\Delta \\ 0, & \text{else.} \end{cases} \tag{3.5}$$

<u>Window Four (Hamming)</u>

$$w_4(x) = \begin{cases} 0.54 + 0.46\cos(\pi x/k\Delta), & |x| \le k\Delta \\ 0, & \text{else.} \end{cases} \tag{3.6}$$

<u>Window Five (Blackman)</u>

$$w_5(x) = \begin{cases} 0.42 + 0.5\cos(\pi x/k\Delta) + 0.08\cos(2\pi x/k\Delta), & |x| \le k\Delta \\ 0, & \text{else.} \end{cases} \tag{3.7}$$

The PSF $h_2(n)$ is simlar to $h_1(n)$, but without a window and scaled differently.

$$h_2(n) = \frac{D}{2d}\Delta h_c(n\Delta), \quad |n| < \infty. \tag{3.8}$$

We point out without proof that the Fourier transform of $h_1(n)$, evaluated at the origin, is equal to $2d/D$, whereas $h_2(n)$ is normalized so that its Fourier transform is unity at the origin for all values of d/D.

All results in this report are for a sample internal Δ of

$$\Delta = \tfrac{1}{4}\frac{\lambda}{D} \tag{3.9}$$

The Fourier transform of the PSF given by $h_1(n)$ for each of the windows and various values of k and d/D is shown in Fig.'s 3.1 to 3.7.

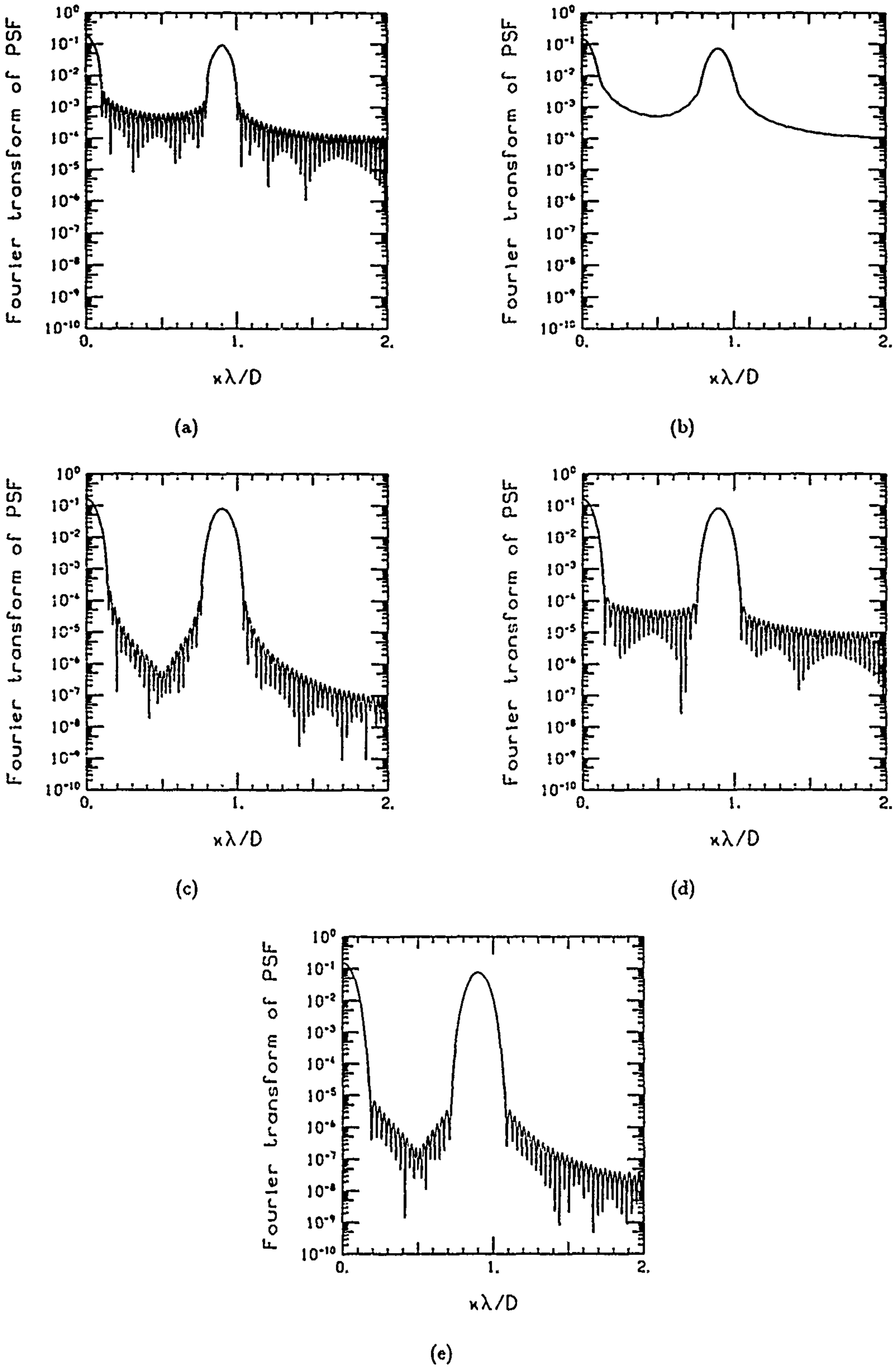


Figure 3.1. Fourier transform of the PSF of Eq. (3.1), $k = 63, d/D = 0.1$. Figures (a)-(e) correspond to window number 1—5, respectively.

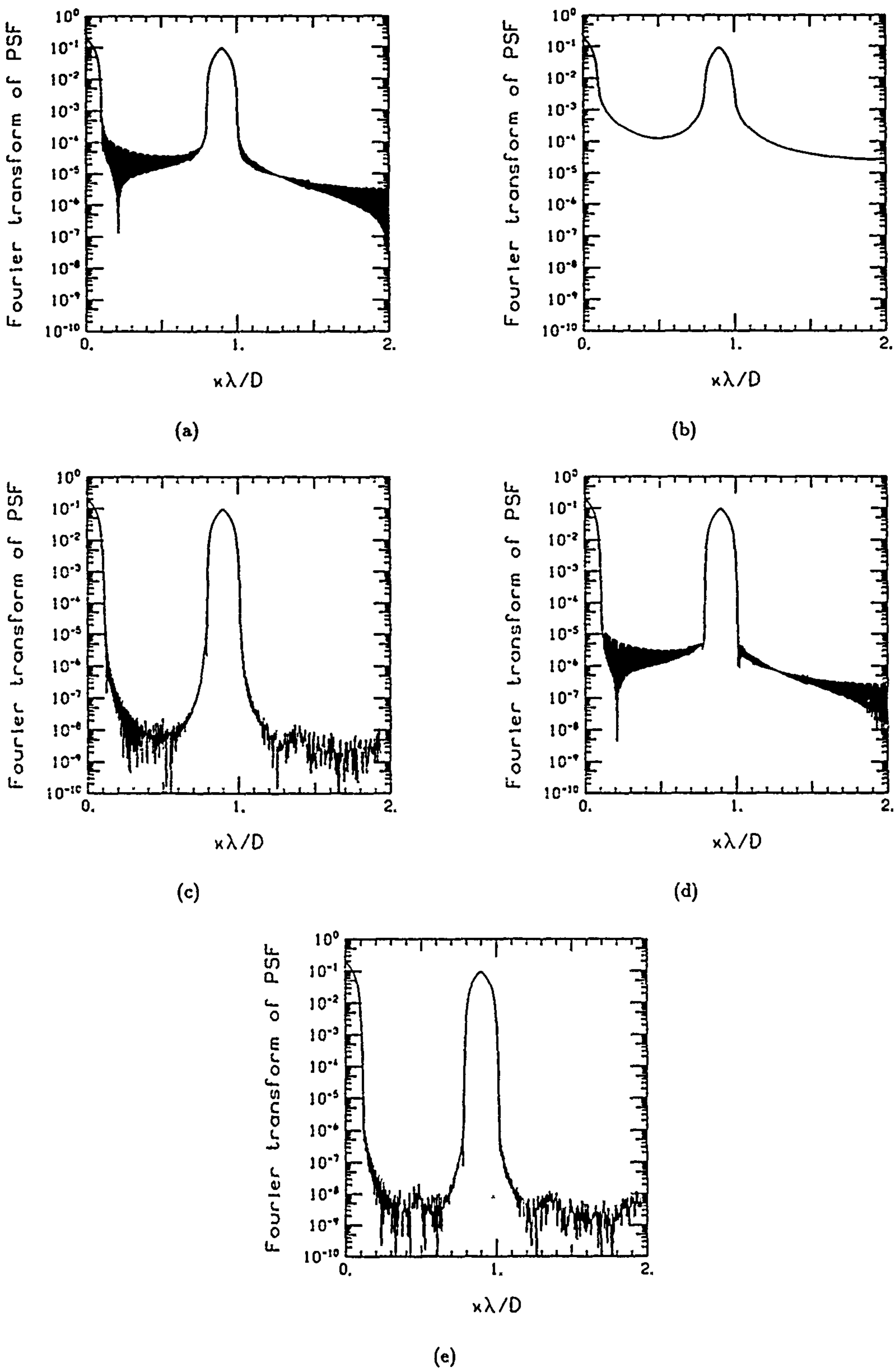


Figure 3.2. Fouier Transform the PSF of Eq. (3.1), $k = 255, d/D = 0.1$. Figures (a)-(e) correspond to window number 1-5, respectively.

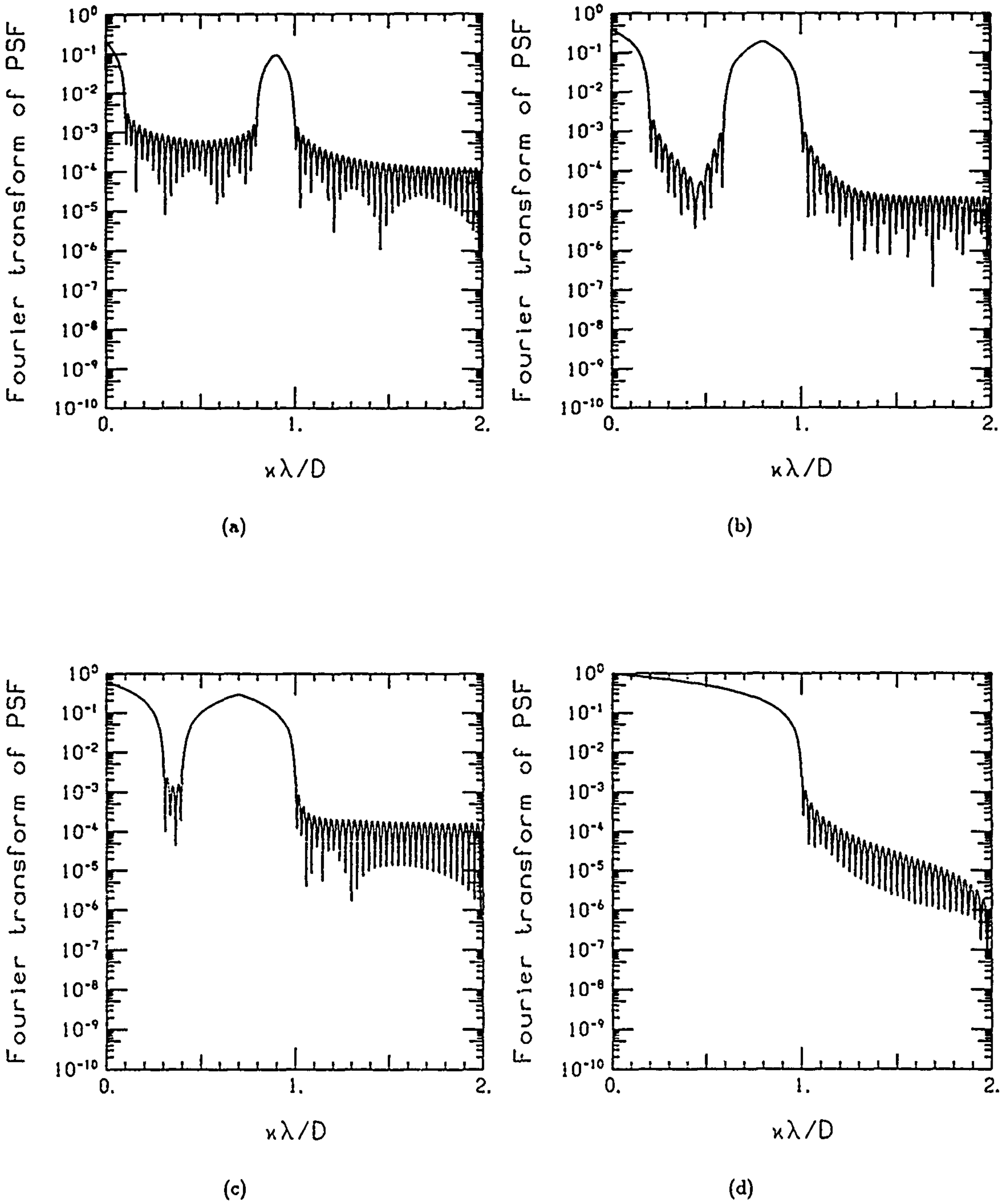


Figure 3.3. Fourier transforms of the PSF of Eq. (3.1), $k = 63$, window number 1. Figures (a)–(d) correspond to $d/D = 0.1, 0.2, 0.3$, and 0.5, respectively.

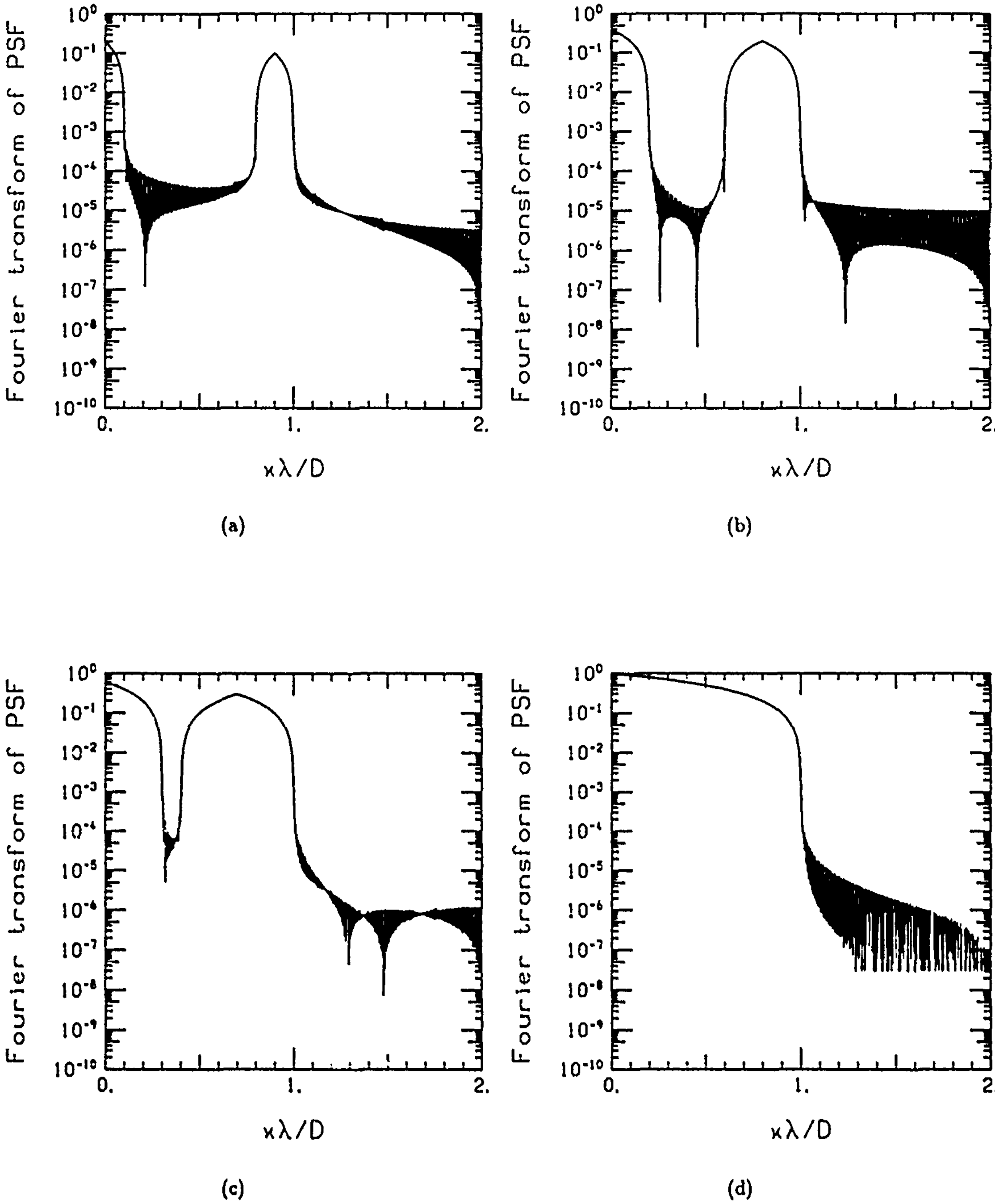


Figure 3.4. Fourier transforms of the PSF of Eq. (3.1), $k = 255$, window number 1. Figures (a)-(d) correspond to $d/D = 0.1, 0.2, 0.3$, and 0.5, respectively.

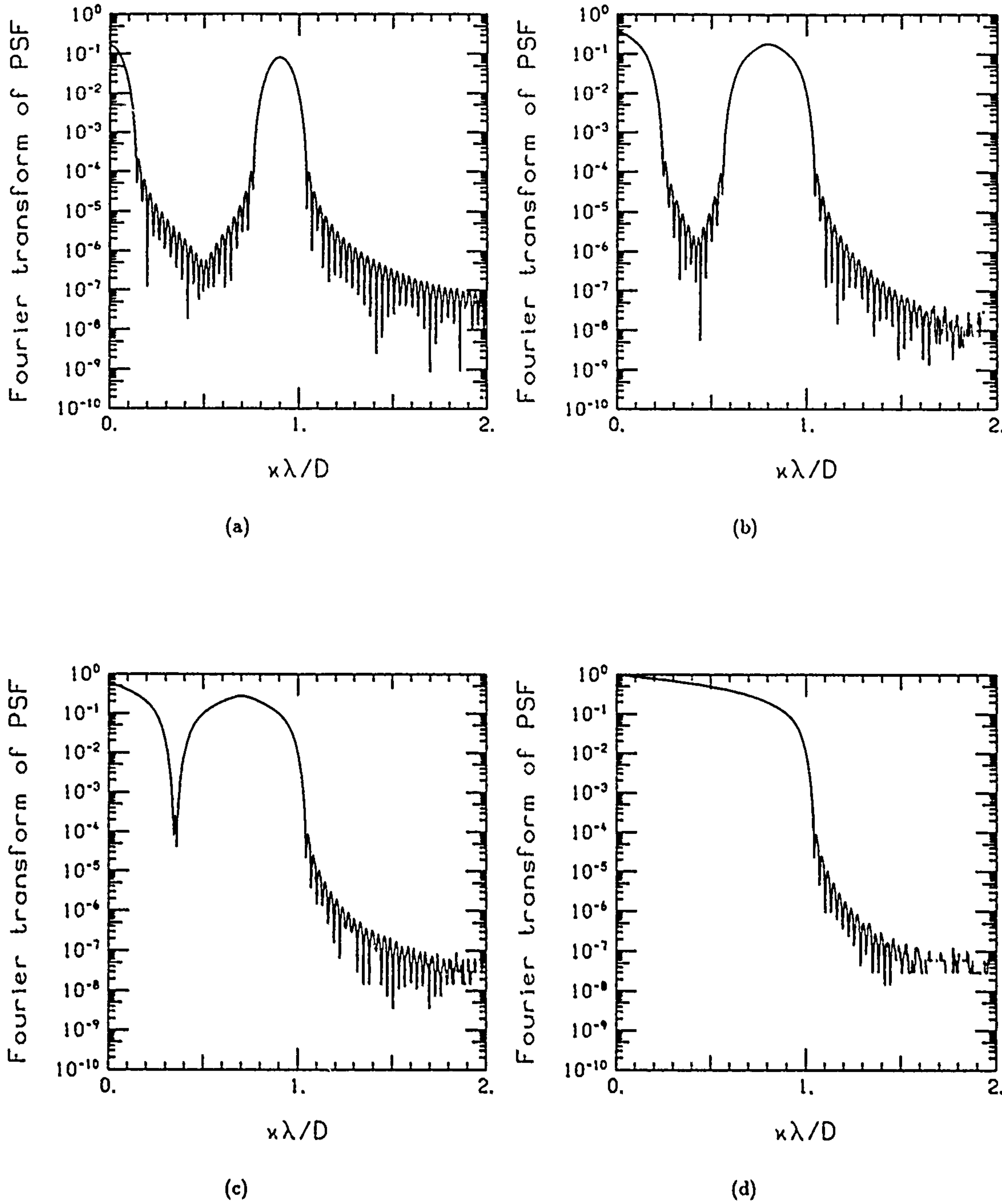


Figure 3.5. Fourier transforms of the PSF of Eq. (3.1), $k = 63$, window number 3. Figures (a)-(d) correspond to $d/D = 0.1, 0.2, 0.3$, and 0.5, respectively.

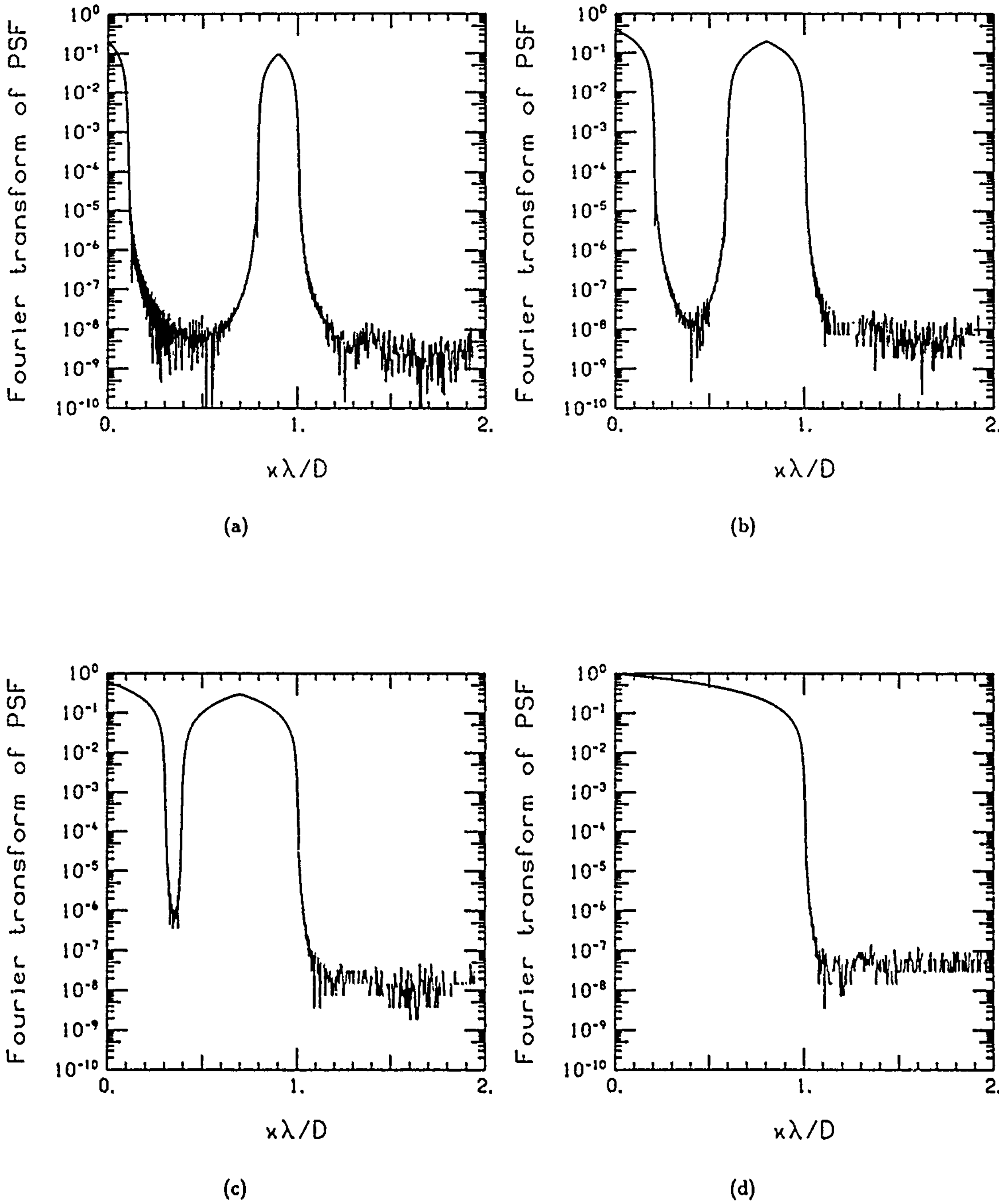


Figure 3.6. Fourier transforms of the PSF of Eq. (3.1), $k = 255$, window number 3. Figures (a)-(d) correspond to $d/D = 0.1, 0.2, 0.3$, and 0.5, respectively.

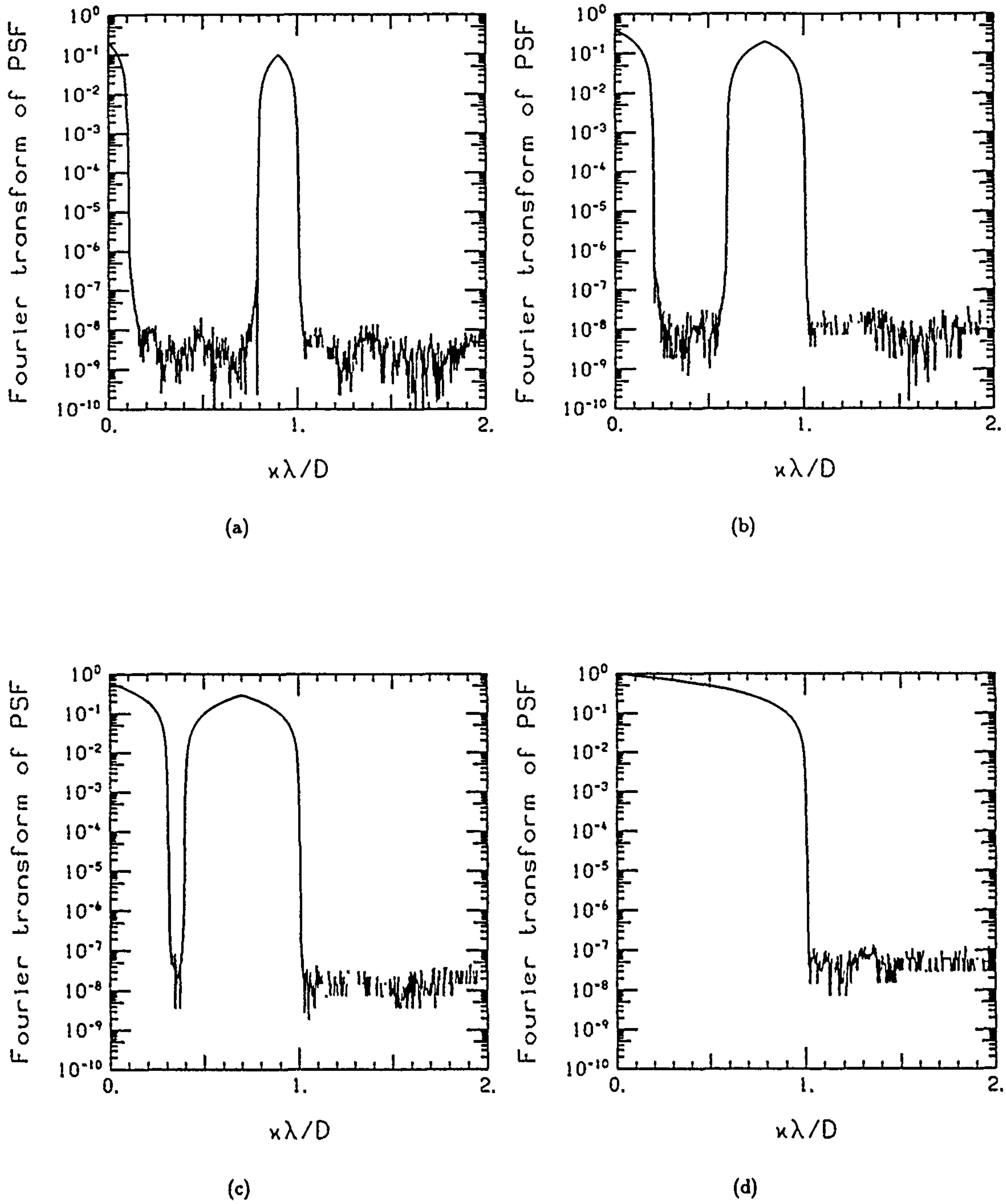


Figure 3.7. Fourier transforms of the PSF of Eq. (3.1), $k = 511$, window number 5. Figures (a)-(d) correspond to $d/D = 0.1, 0.2, 0.3$, and 0.5, respectively.

3.3. Results Using the Minimum-Variance Method

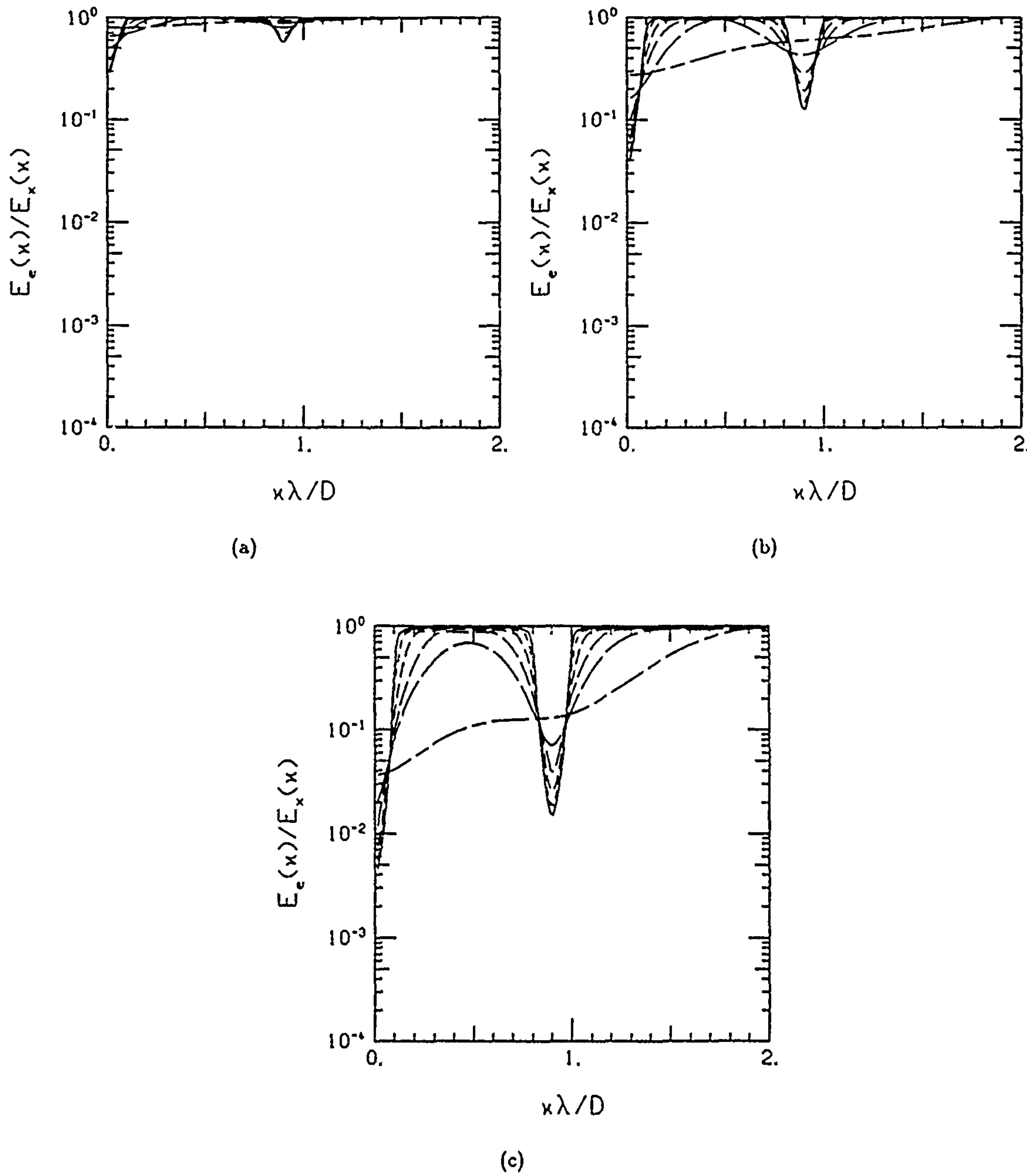


Figure 3.8. Energy spectra ratios using the PSF of Eq. (3.1) with window number three, $k = 255$ (corresponding to an image line length of 638 pixels or $159.5\lambda/D$), and $d/D = 0.1$. Figures (a)-(c) correspond to SNR_{REF} of 20 dB, 30 dB, and 40 dB, respectively. The six curves in each figure correspond to object supports of 32 λ/D (solid curve), $16\lambda/D, 8\lambda/D, 4\lambda/D, 2\lambda/D$, and $1\lambda/D$.

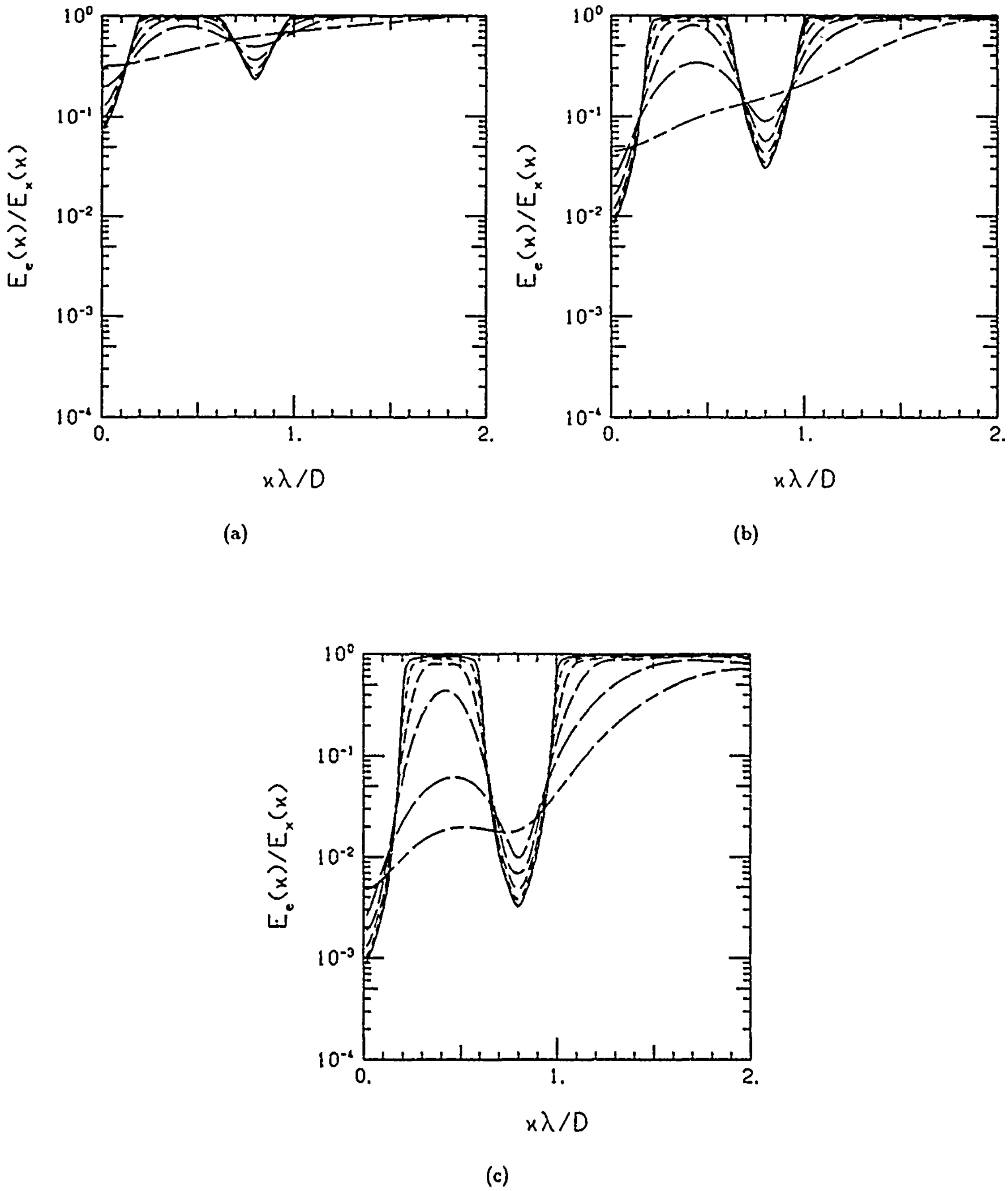


Figure 3.9. Energy spectra ratios using the PSF of Eq. (3.1) with window number three, $k = 255$ (corresponding to an image line length of 638 pixels or $159.5\lambda/D$), and $d/D = 0.2$. Figures (a)–(c) correspond to $\mathrm{SNR_{REF}}$ of 20 dB, 30 dB, and 40 dB, respectively . The six curves in each figure correspond to object supports of 32 λ/D (solid curve), $16\lambda/D, 8\lambda/D, 4\lambda/D, 2\lambda/D$, and $1\lambda/D$.

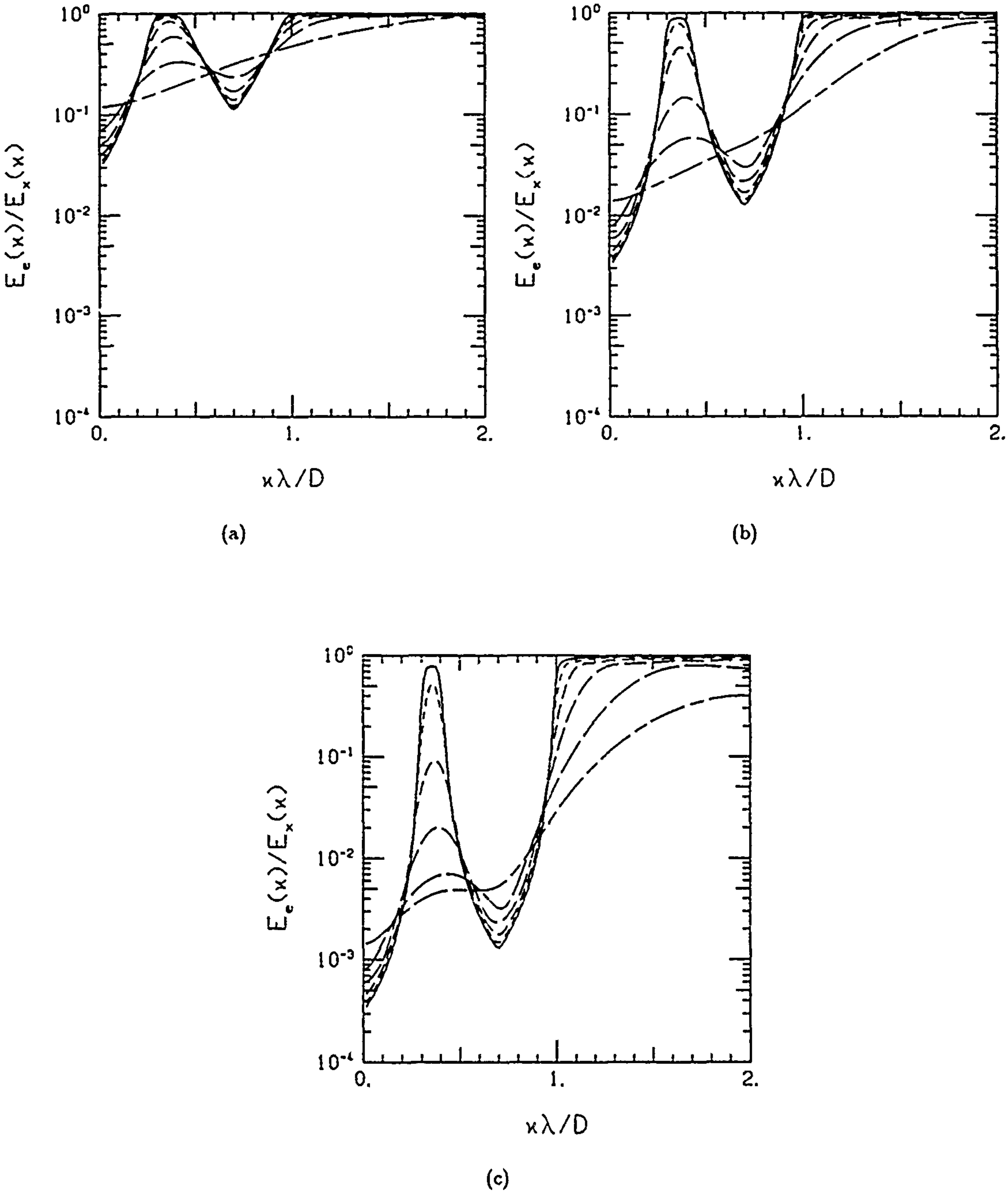


Figure 3.10. Energy spectra ratios using the PSF of Eq. (3.1) with window number three, $k = 255$ (corresponding to an image line length of 638 pixels or $159.5\lambda/D$), and $d/D = 0.3$. Figures (a)-(c) correspond to SNR_{REF} of 20 dB, 30 dB, and 40 dB, respectively. The six curves in each figure correspond to object supports of 32 λ/D (solid curve), $16\lambda/D, 8\lambda/D, 4\lambda/D, 2\lambda/D$, and $1\lambda/D$.

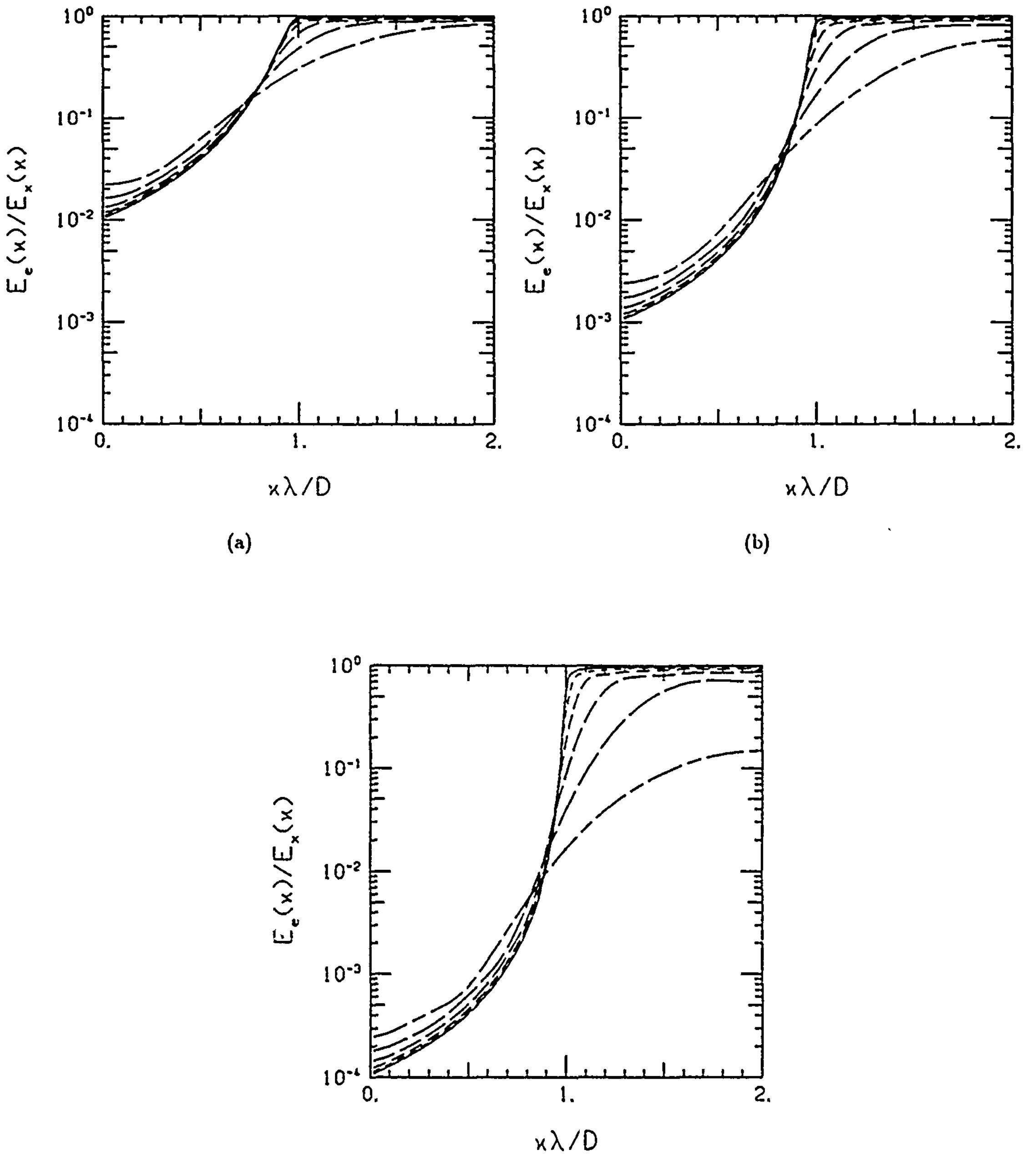


Figure 3.11. Energy spectra ratios using the PSF of Eq. (3.1), with window number three, $k = 255$ (corresponding to an image line length of 638 pixels or $159.5\lambda/D$), and $d/D = 0.5$. Figures (a)–(c) correspond to SNR_{REF} of 20 dB, 30 dB, and 40 dB, respectively. The six curves in each figure correspond to object supports of 32 λ/D (solid curve), $16\lambda/D, 8\lambda/D, 4\lambda/D, 2\lambda/D$, and $1\lambda/D$.

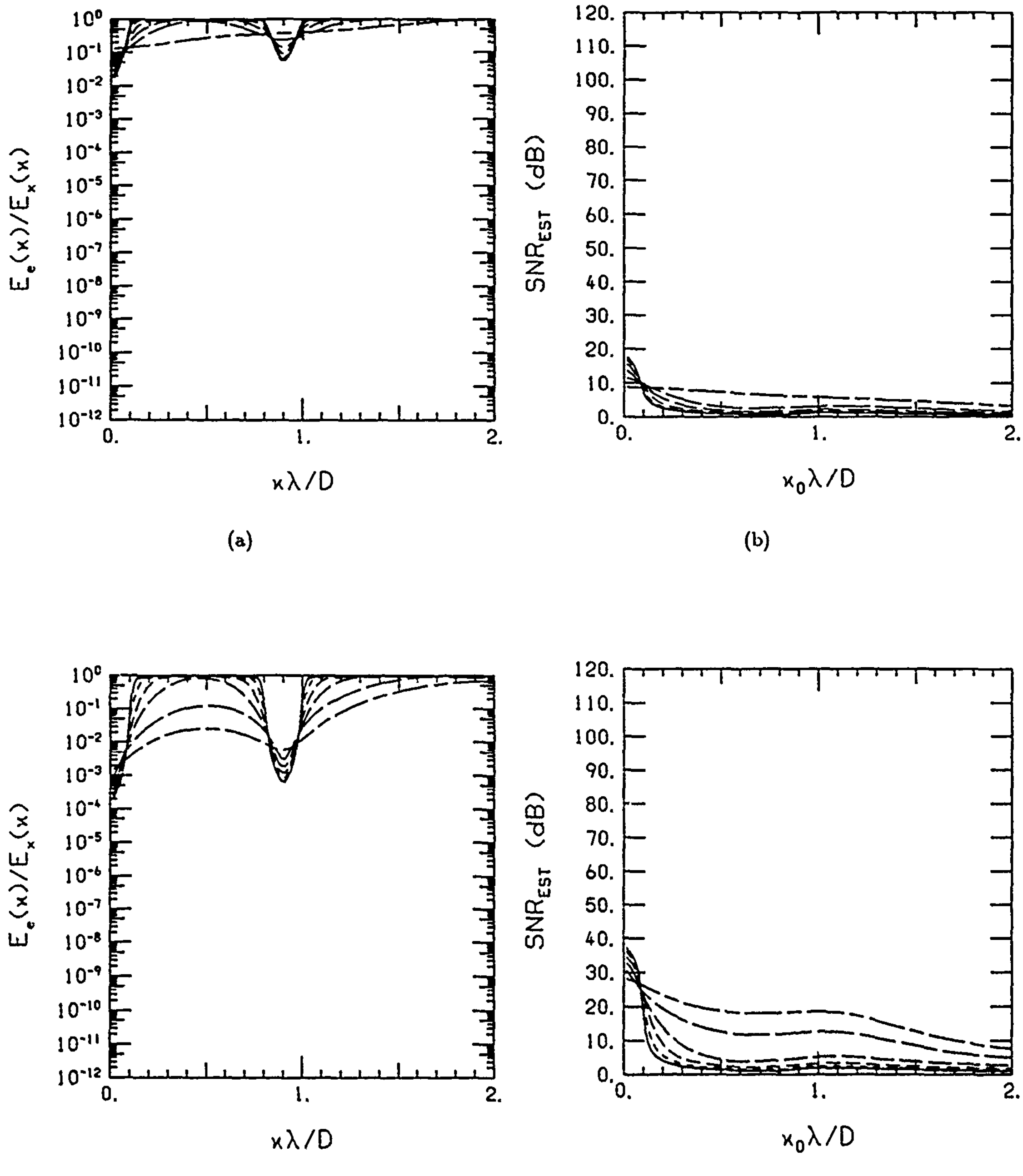
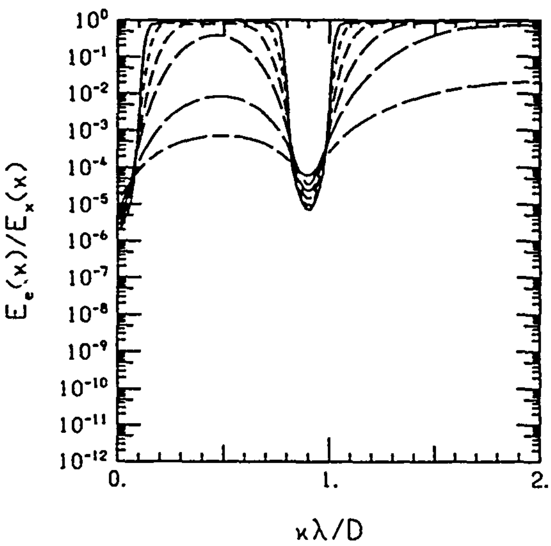
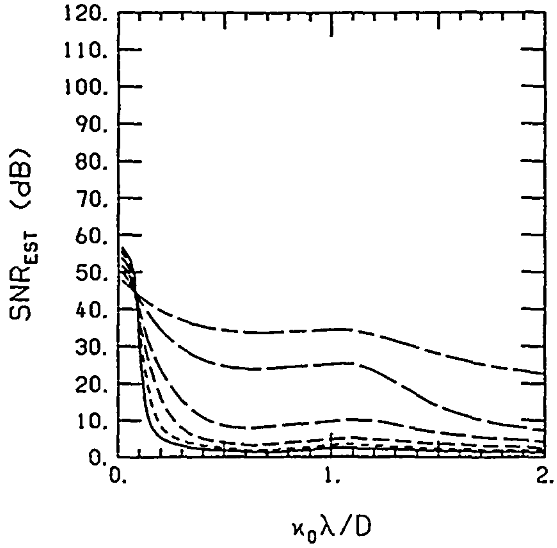


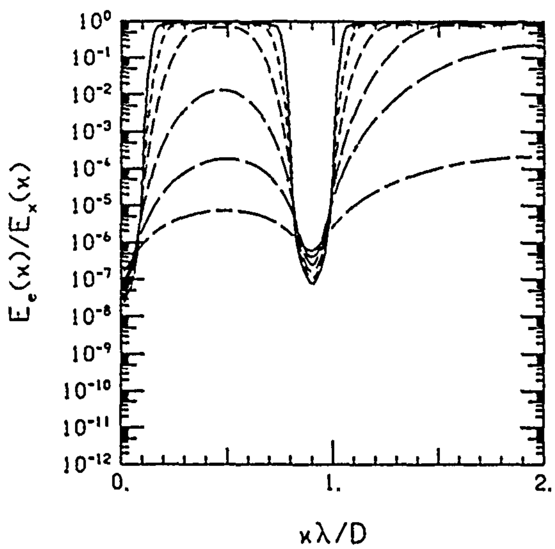
Figure 3.12. Energy spectra ratios and SNR_{EST} using the PSF of Eq. (3.8), an image line length of 1024 pixels ($256\lambda/D$), and $d/D = 0.1$. The six curves of each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D, 8\lambda/D, 4\lambda/D, 2\lambda/D$, and $1\lambda/D$. Figures (a) and (b) correspond to SNR_{REF}=20 dB and figures (c) and (d) correspond to SNR_{REF}=40 dB.



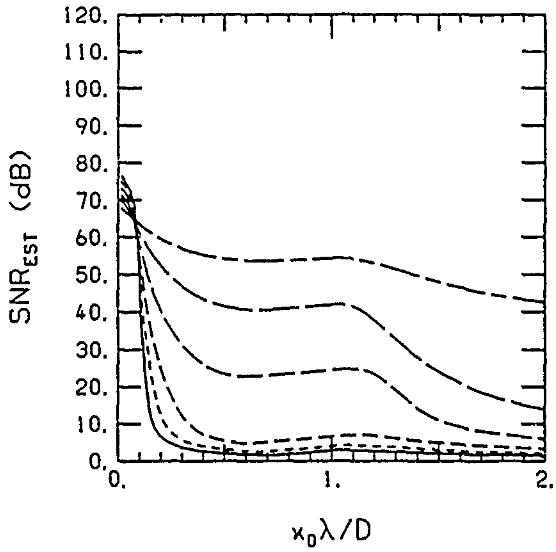
(a)



(b)



(c)



(d)

Figure 3.13. Energy spectra ratios and SNR_{EST} using the PSF of Eq (3.8), an image line length of 1024 pixels ($256\lambda/D$), and $d/D = 0.1$. The six curves of each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D, 8\lambda/D, 4\lambda/D, 2\lambda/D$, and $1\lambda/D$. Figures (a) and (b) correspond to SNR_{REF}=60 dB and figures (c) and (d) correspond to SNR_{REF}=80 dB.

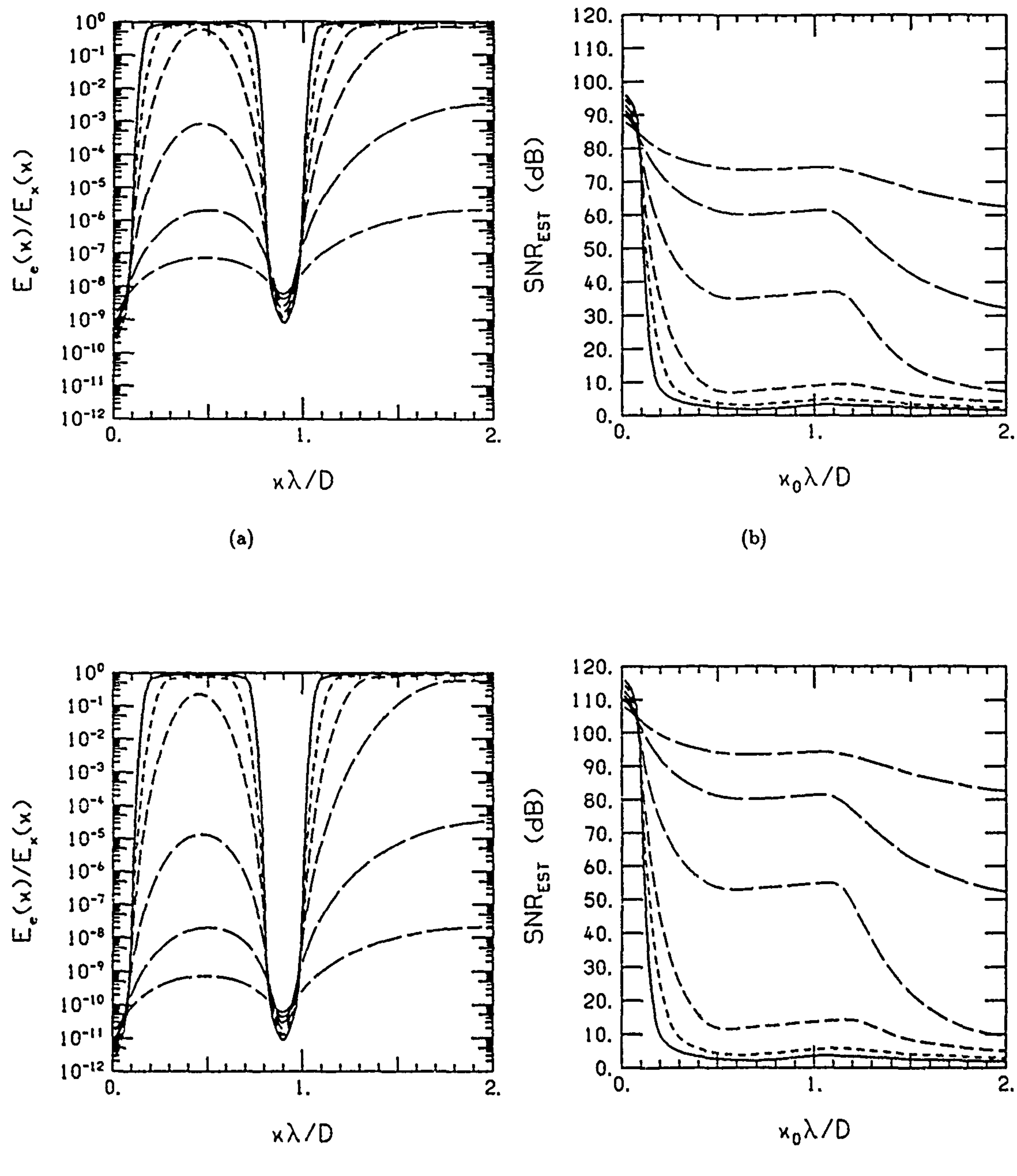


Figure 3.14. Energy spectra ratios and SNR_{EST} using the PSF of Eq (3.8), an image line length of 1024 pixels ($256\lambda/D$), and $d/D = 0.1$. The six curves of each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D, 8\lambda/D, 4\lambda/D, 2\lambda/D$, and $1\lambda/D$. Figures (a) and (b) correspond to $SNR_{REF} = 100dB$ and figures (c) and (d) correspond to SNR_{REF}=120 dB.

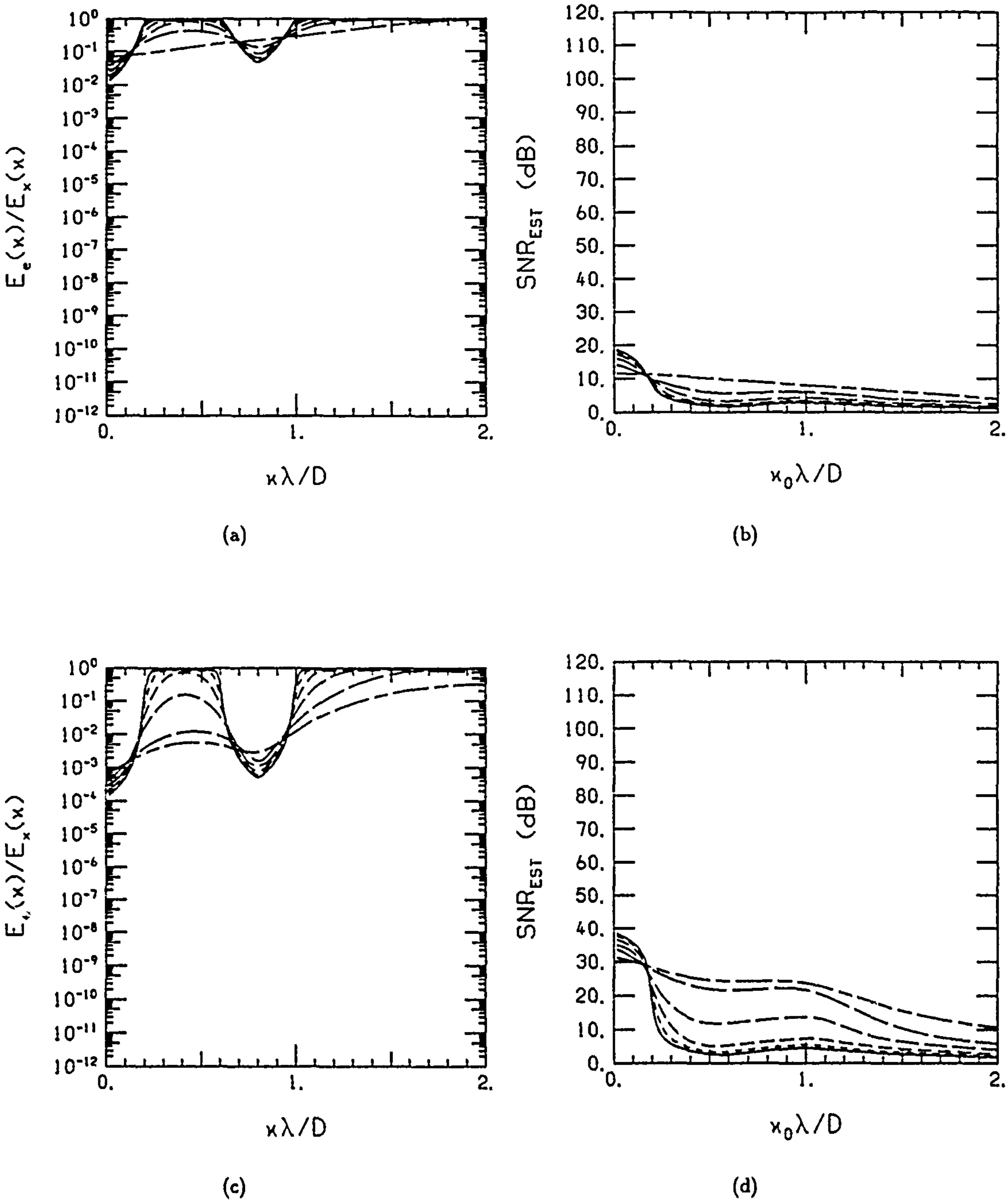


Figure 3.15. Energy spectra ratios and SNR_{EST} using the PSF of Eq (3.8), an image line length of 1024 pixels ($256\lambda/D$), and $d/D = 0.2$. The six curves of each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D, 8\lambda/D, 4\lambda/D, 2\lambda/D$, and $1\lambda/D$. Figures (a) and (b) correspond to $SNR_{REF} = 20dB$ and figures (c) and (d) correspond to SNR_{REF}=40 dB.

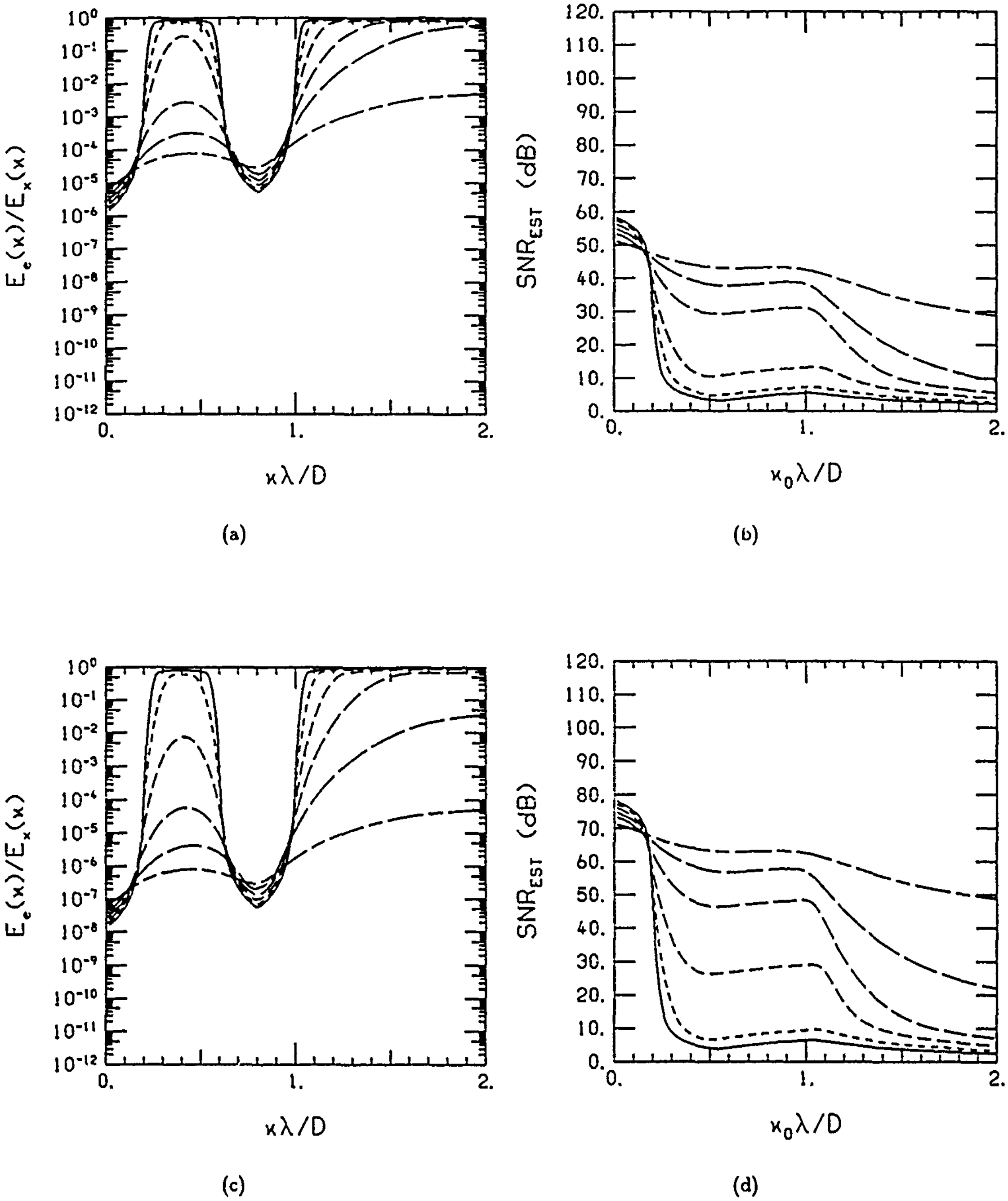
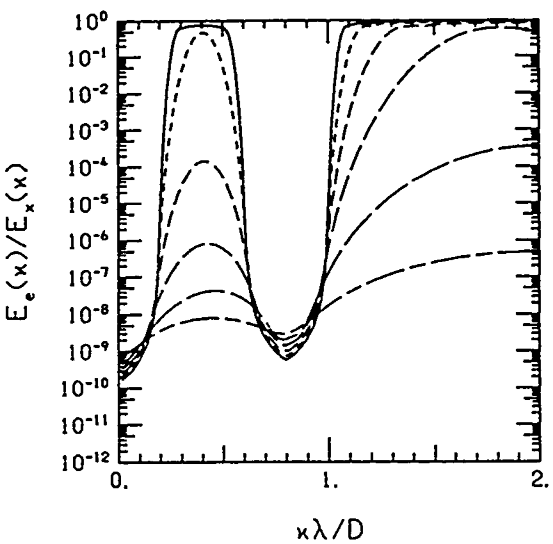
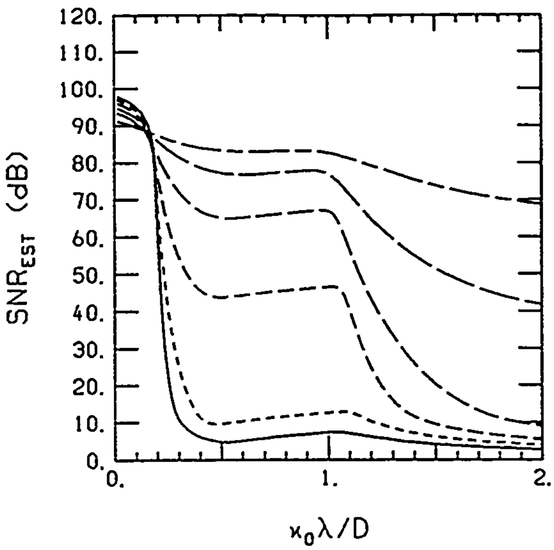


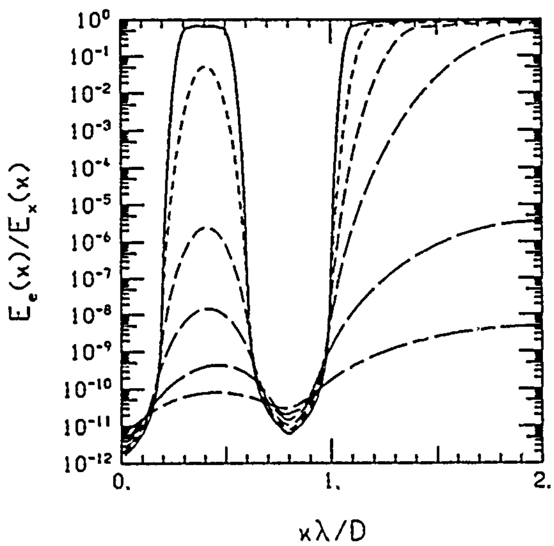
Figure 3.16. Energy spectra ratios and SNR_{EST} using the PSF of Eq (3.8), an image line length of 1024 pixels ($256\lambda/D$), and $d/D = 0.2$. The six curves of each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D, 8\lambda/D, 4\lambda/D, 2\lambda/D$, and $1\lambda/D$. Figures (a) and (b) correspond to $SNR_{REF} = 60dB$ and figures (c) and (d) correspond to SNR_{REF}=80 dB.



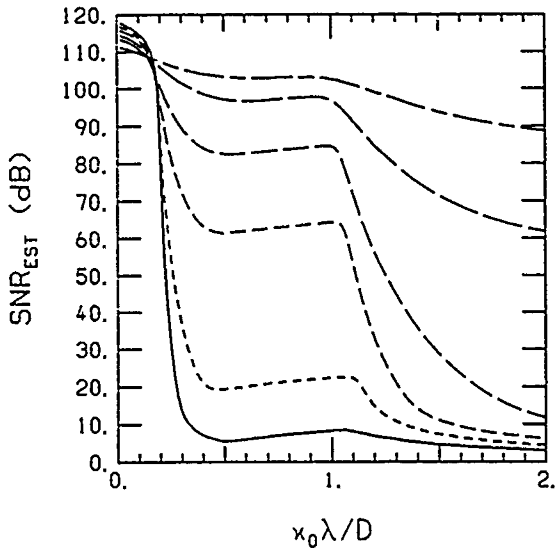
(a)



(b)



(c)



(d)

Figure 3.17. Energy spectra ratios and SNR_{EST} using the PSF of Eq (3.8), an image line length of 1024 pixels ($256\lambda/D$), and $d/D = 0.2$. The six curves of each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D, 8\lambda/D, 4\lambda/D, 2\lambda/D$, and $1\lambda/D$. Figures (a) and (b) correspond to $SNR_{REF} = 100dB$ and figures (c) and (d) correspond to SNR_{REF}=120 dB.

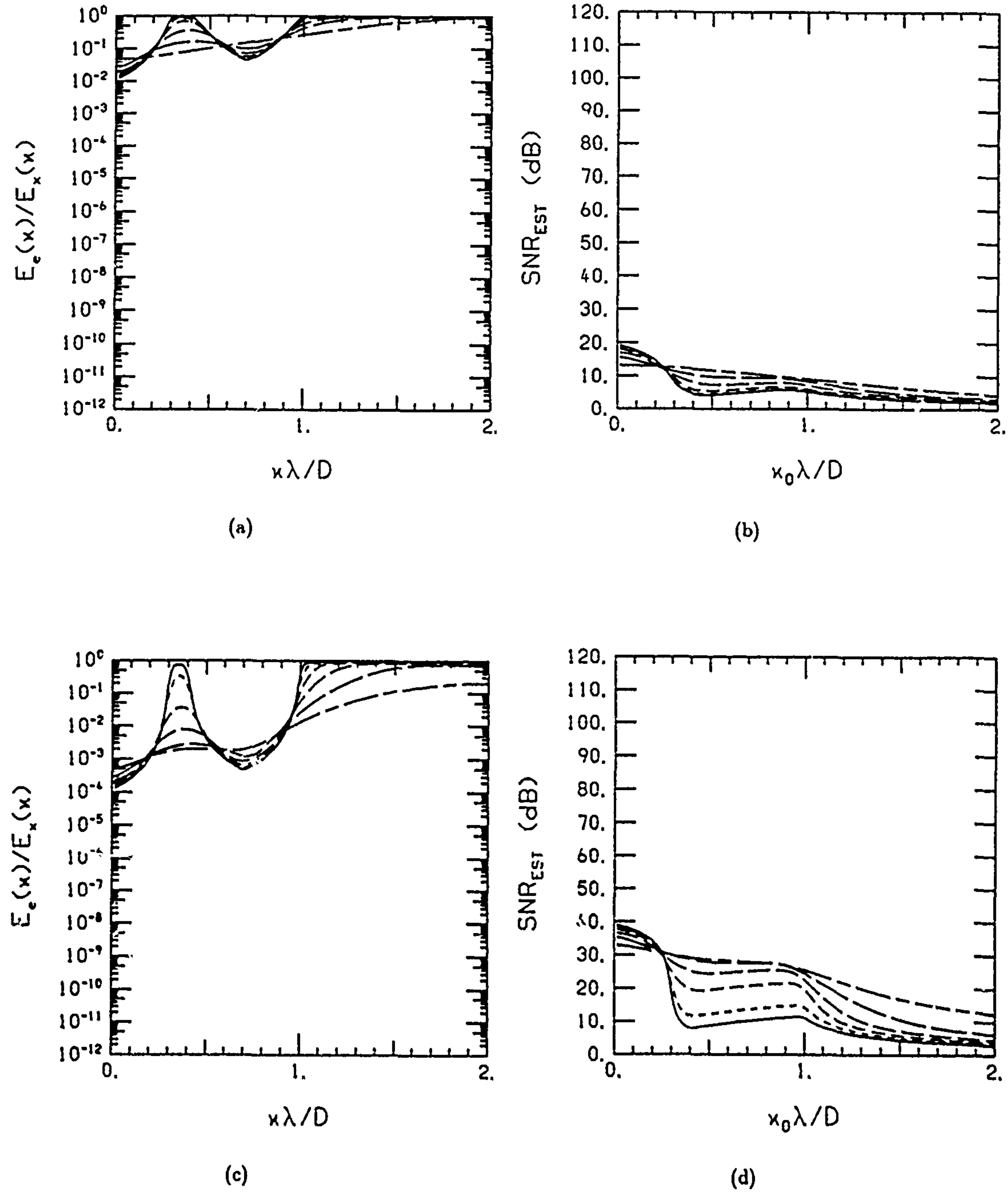


Figure 3.18. Energy spectra ratios and SNR_{EST} using the PSF of Eq (3.8), an image line length of 1024 pixels ($256\lambda/D$), and $d/D = 0.3$. The six curves of each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D, 8\lambda/D, 4\lambda/D, 2\lambda/D$, and $1\lambda/D$. Figures (a) and (b) correspond to $SNR_{REF} = 20dB$ and figures (c) and (d) correspond to SNR_{REF}=40 dB.

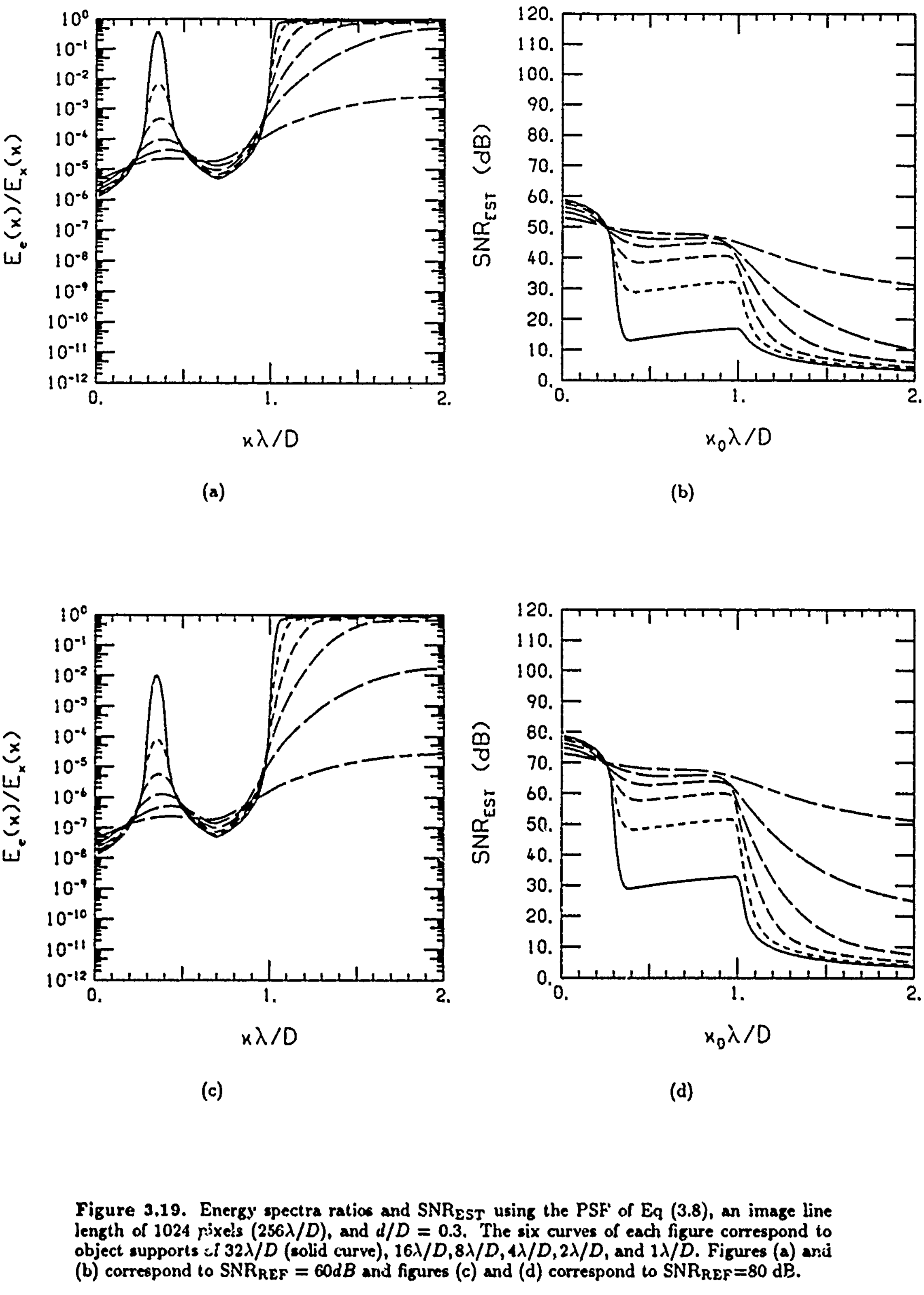


Figure 3.19. Energy spectra ratios and SNR_{EST} using the PSF of Eq (3.8), an image line length of 1024 pixels ($256\lambda/D$), and $d/D = 0.3$. The six curves of each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D, 8\lambda/D, 4\lambda/D, 2\lambda/D$, and $1\lambda/D$. Figures (a) and (b) correspond to $SNR_{REF} = 60dB$ and figures (c) and (d) correspond to SNR_{REF}=80 dB.

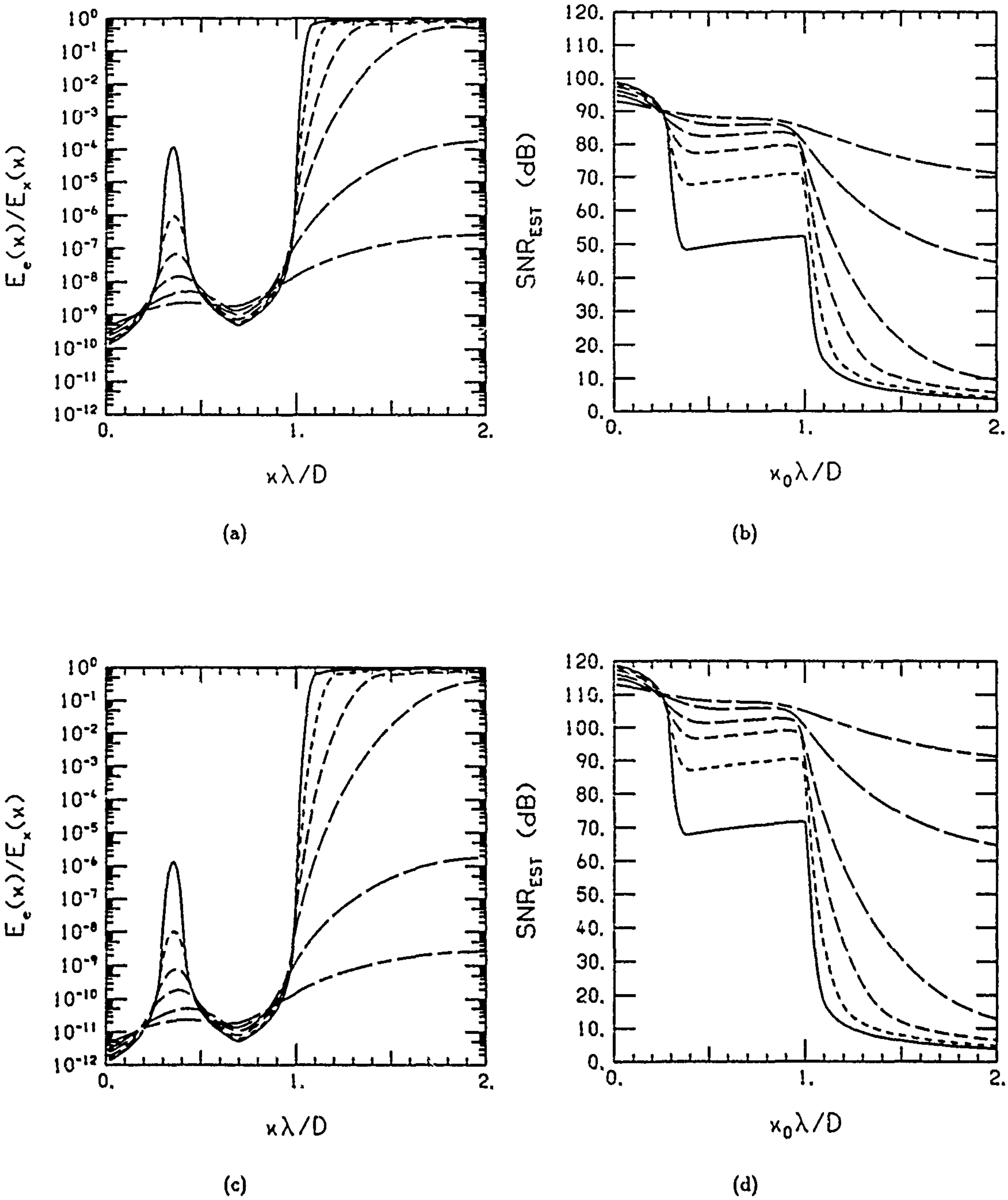


Figure 3.20. Energy spectra ratios and SNR_{EST} using the PSF of Eq (3.8), an image line length of 1024 pixels ($256\lambda/D$), and $d/D = 0.3$. The six curves of each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D, 8\lambda/D, 4\lambda/D, 2\lambda/D$, and $1\lambda/D$. Figures (a) and (b) correspond to $SNR_{REF} = 100dB$ and figures (c) and (d) correspond to SNR_{REF}=120 dB.

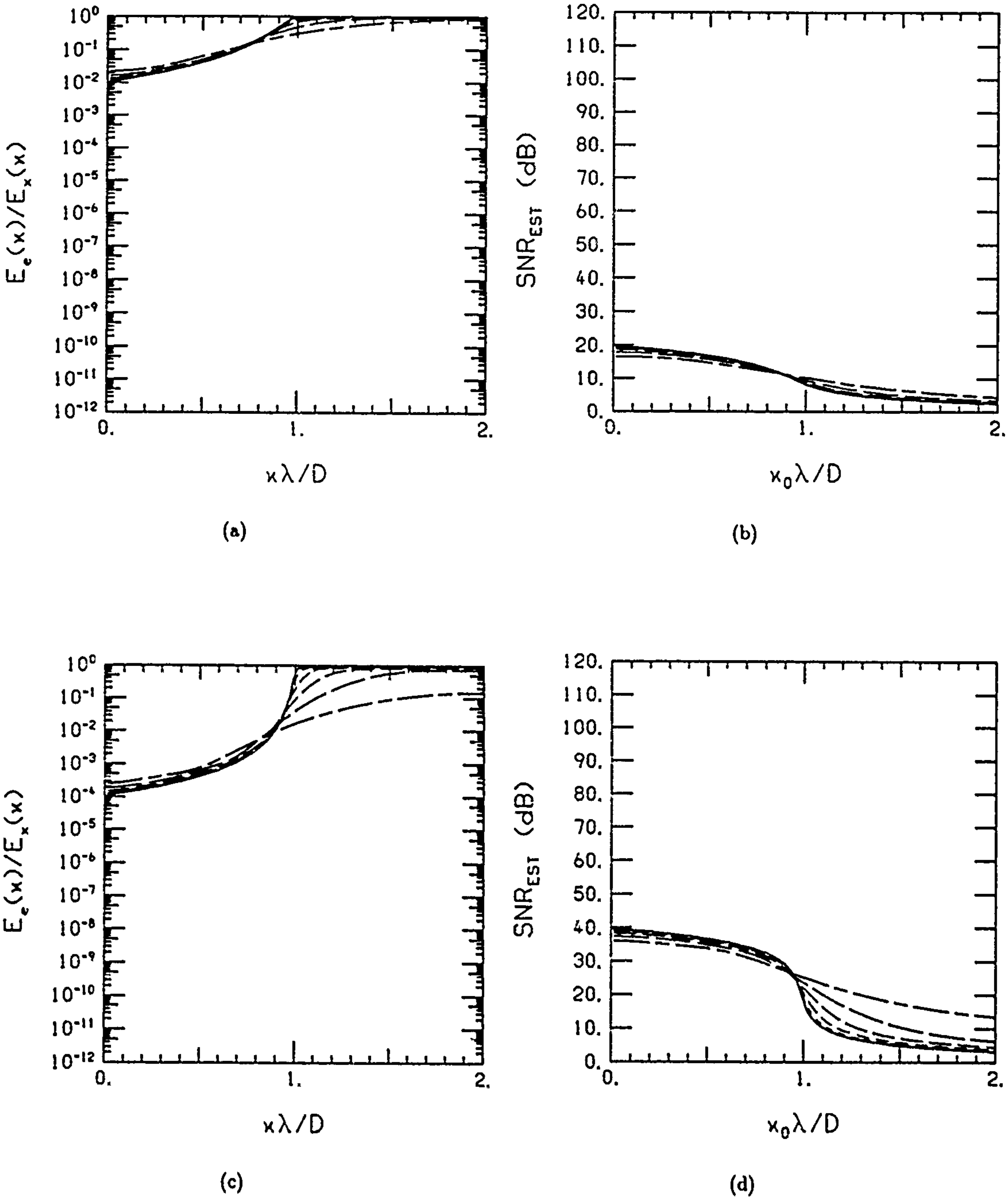


Figure 3.21. Energy spectra ratios and SNR_{EST} using the PSF of Eq (3.8), an image line length of 1024 pixels $(256\lambda/D)$, and $d/D = 0.5$. The six curves of each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D, 8\lambda/D, 4\lambda/D, 2\lambda/D$, and $1\lambda/D$. Figures (a) and (b) correspond to $SNR_{REF} = 20dB$ and figures (c) and (d) correspond to SNR_{REF}=40 dB.

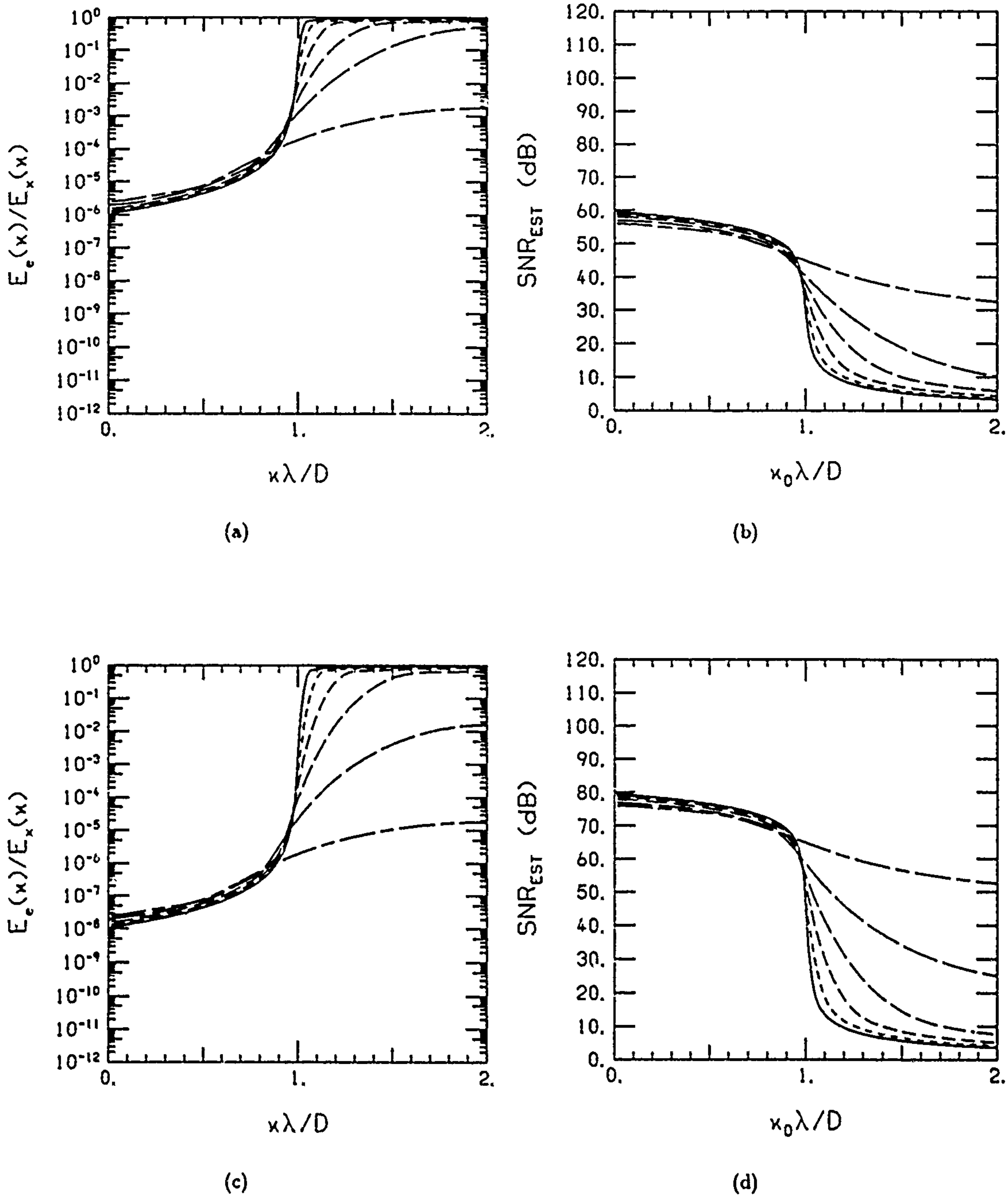


Figure 3.22. Energy spectra ratios and SNR_{EST} using the PSF of Eq (3.8), an image line length of 1024 pixels ($256\lambda/D$), and $d/D = 0.5$. The six curves of each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D, 8\lambda/D, 4\lambda/D, 2\lambda/D$, and $1\lambda/D$. Figures (a) and (b) correspond to $SNR_{REF} = 60dB$ and figures (c) and (d) correspond to SNR_{REF}=80 dB.

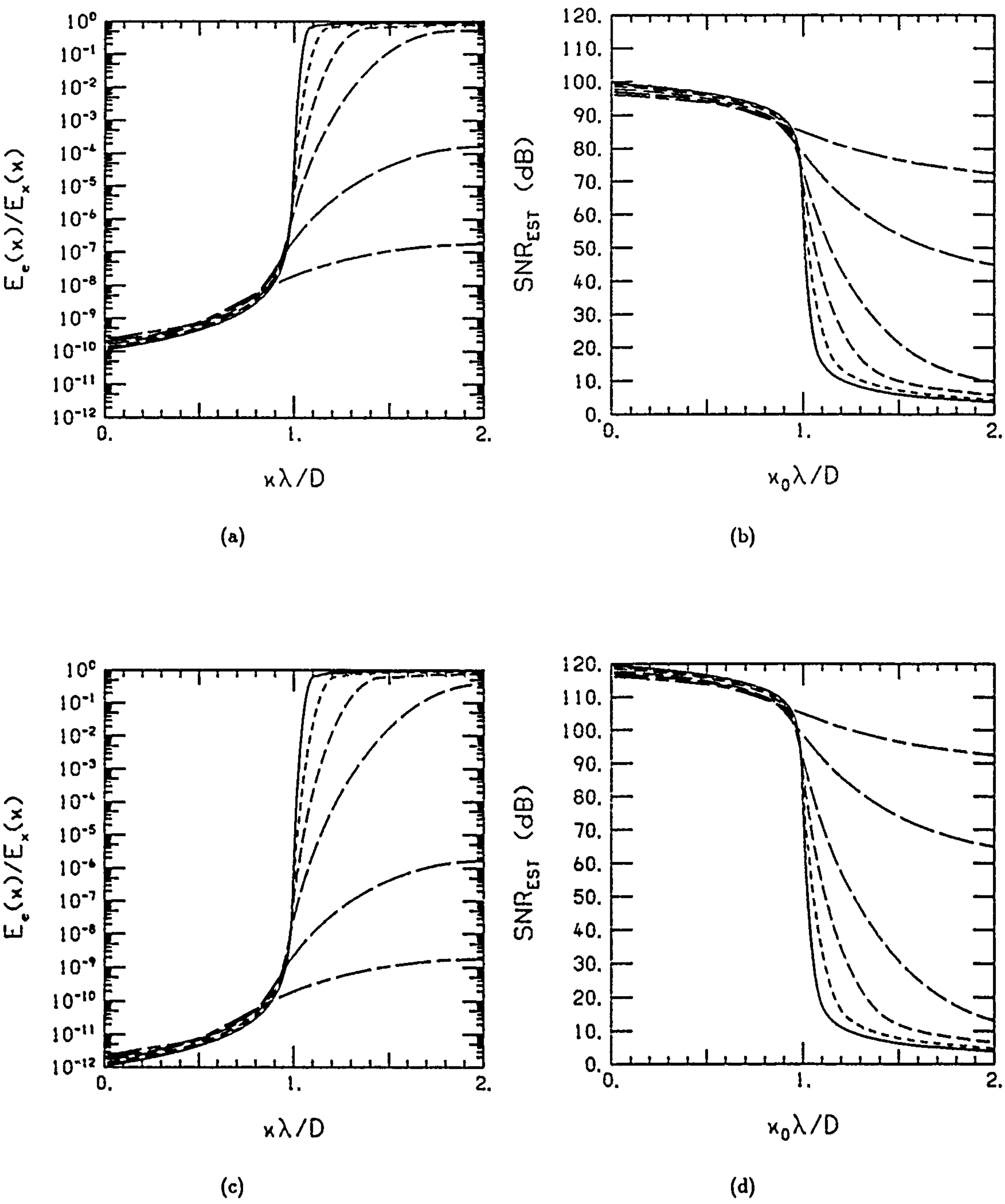


Figure 3.23. Energy spectra ratios and SNR_{EST} using the PSF of Eq (3.8), an image line length of 1024 pixels ($256\lambda/D$), and $d/D = 0.5$. The six curves of each figure correspond to object supports of $32\lambda/D$ (solid curve), $16\lambda/D$, $8\lambda/D$, $4\lambda/D$, $2\lambda/D$, and $1\lambda/D$. Figures (a) and (b) correspond to $SNR_{REF} = 100$ dB and figures (c) and (d) correspond to SNR_{REF}=120 dB.

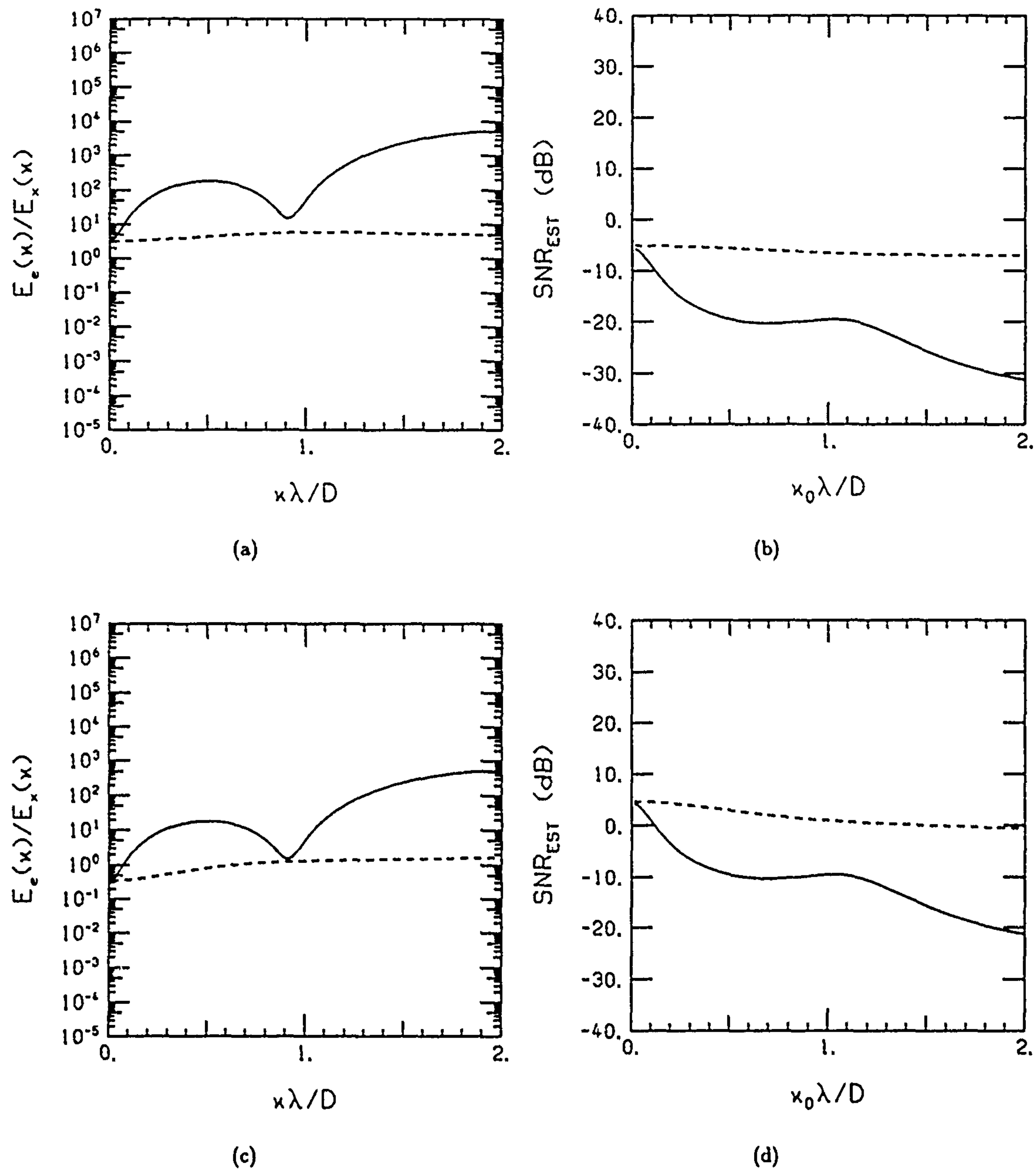


Figure 3.24. Energy spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number one and $d/D = 0.1, k = 63$ (corresponding to an image line length of 254 pixels or $63.5\lambda/D$), and an object support of 1 λ/D. Figures (a) and (b) correspond to SNR_{REF}=20 dB, and figures (c) and (d) correspond to SNR_{REF}=30 dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.

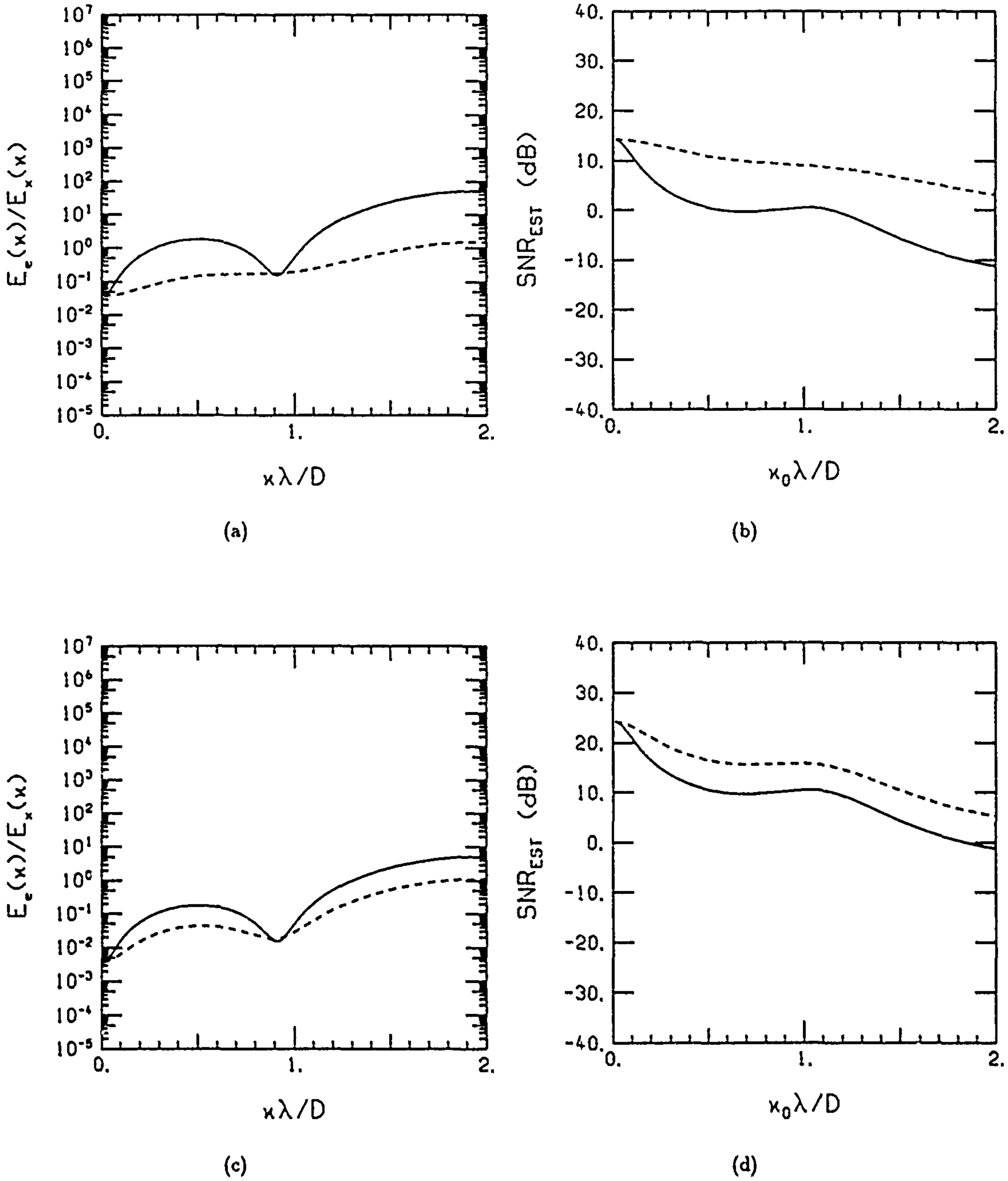


Figure 3.25. Energy spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number one and $d/D = 0.1, k = 63$ (corresponding to an image line length of 254 pixels or $63.5\lambda/D$), and an object support of 1 λ/D. Figures (a) and (b) correspond to SNR_{REF}=40 dB, and figures (c) and (d) correspond to SNR_{REF}=50 dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.

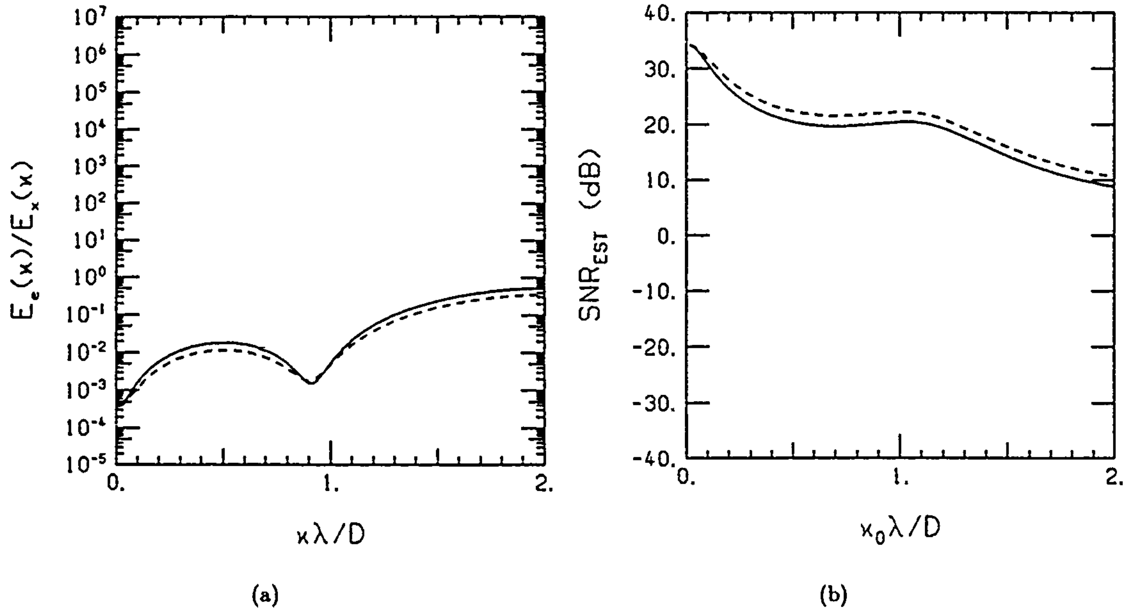


Figure 3.26. Energy Spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number one and $d/D = 0.1, k = 63$ (corresponding to an image line length of 254 pixels or $63.5\lambda/D$), an object support of 1 λ/D, and SNR_{REF}=60 dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.

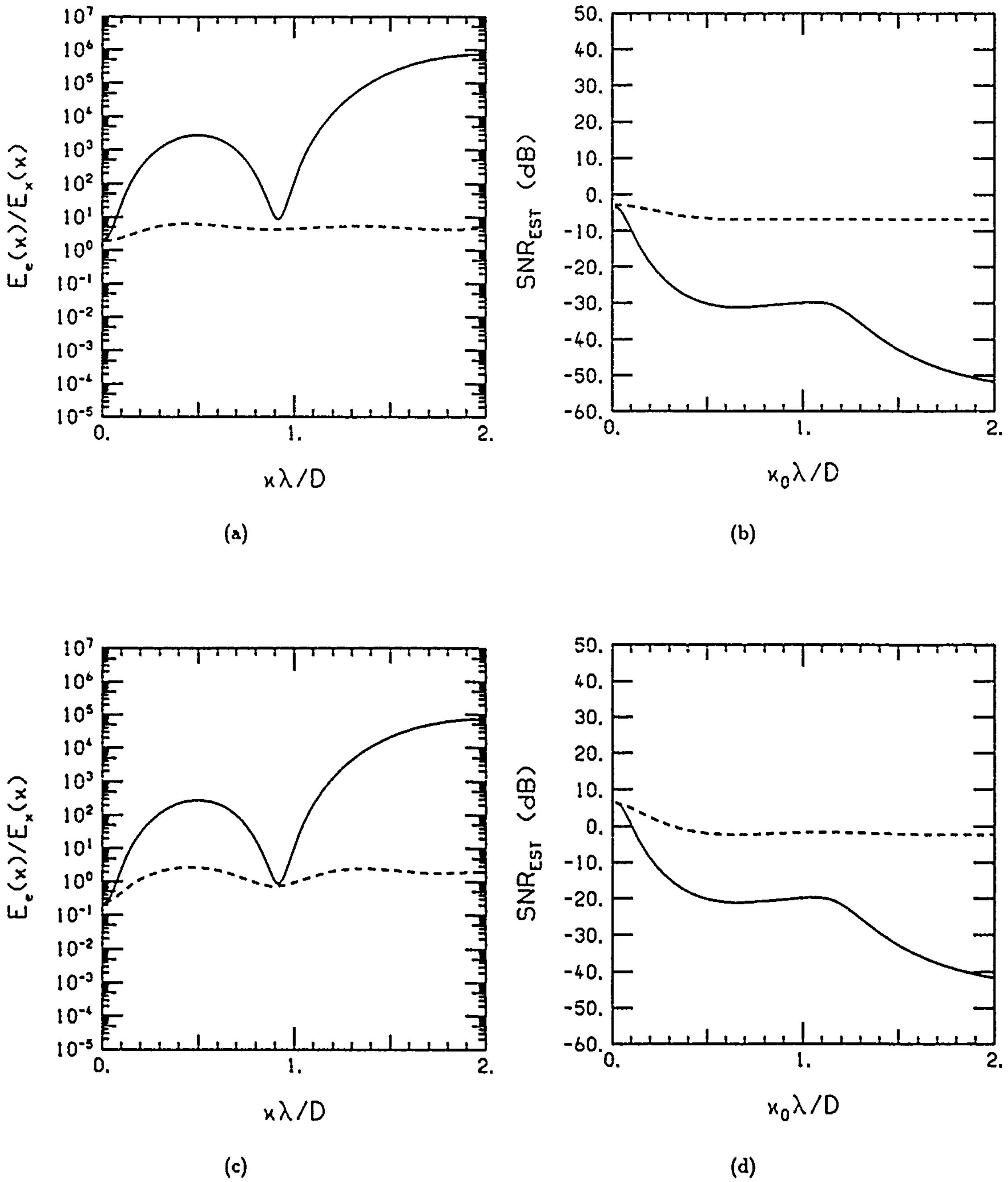


Figure 3.27. Energy spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number one and $d/D = 0.1, k = 63$ (corresponding to an image line length of 254 pixels or $63.5\lambda/D$), and an object support of 2 λ/D. Figures (a) and (b) correspond to SNR_{REF}=20 dB, and figures (c) and (d) correspond to SNR_{REF}=30 dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.

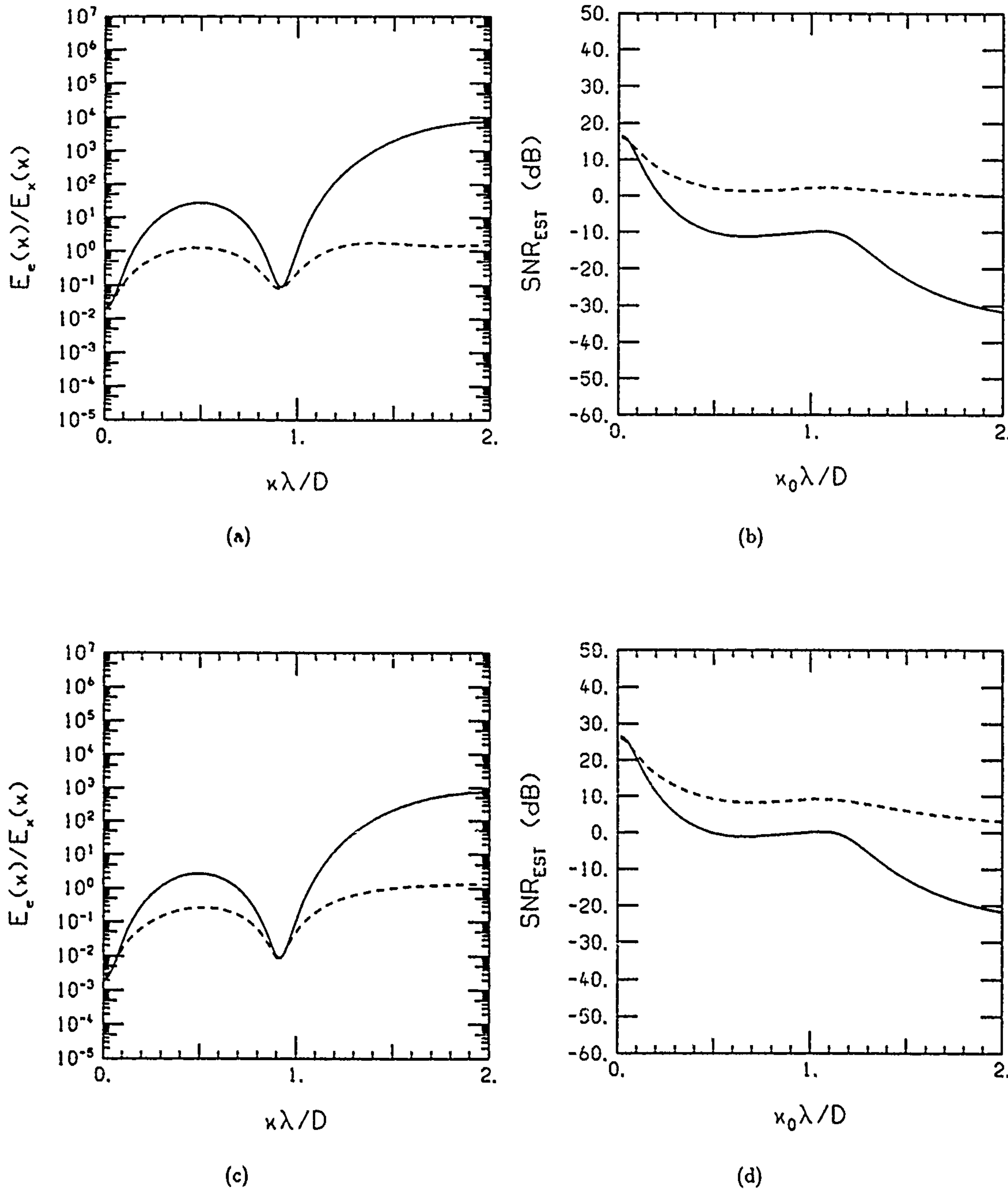


Figure 3.28. Energy spectra ratios and $\mathrm{SNR_{EST}}$ using the PSF of Eq. (3.1) with window number one and $d/D = 0.1, k = 63$ (corresponding to an image line length of 254 pixels or $63.5\lambda/D$), and an object support of 2 λ/D. Figures (a) and (b) correspond to $\mathrm{SNR_{REF}}$=40 dB, and figures (c) and (d) correspond to $\mathrm{SNR_{REF}}$=50 dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.

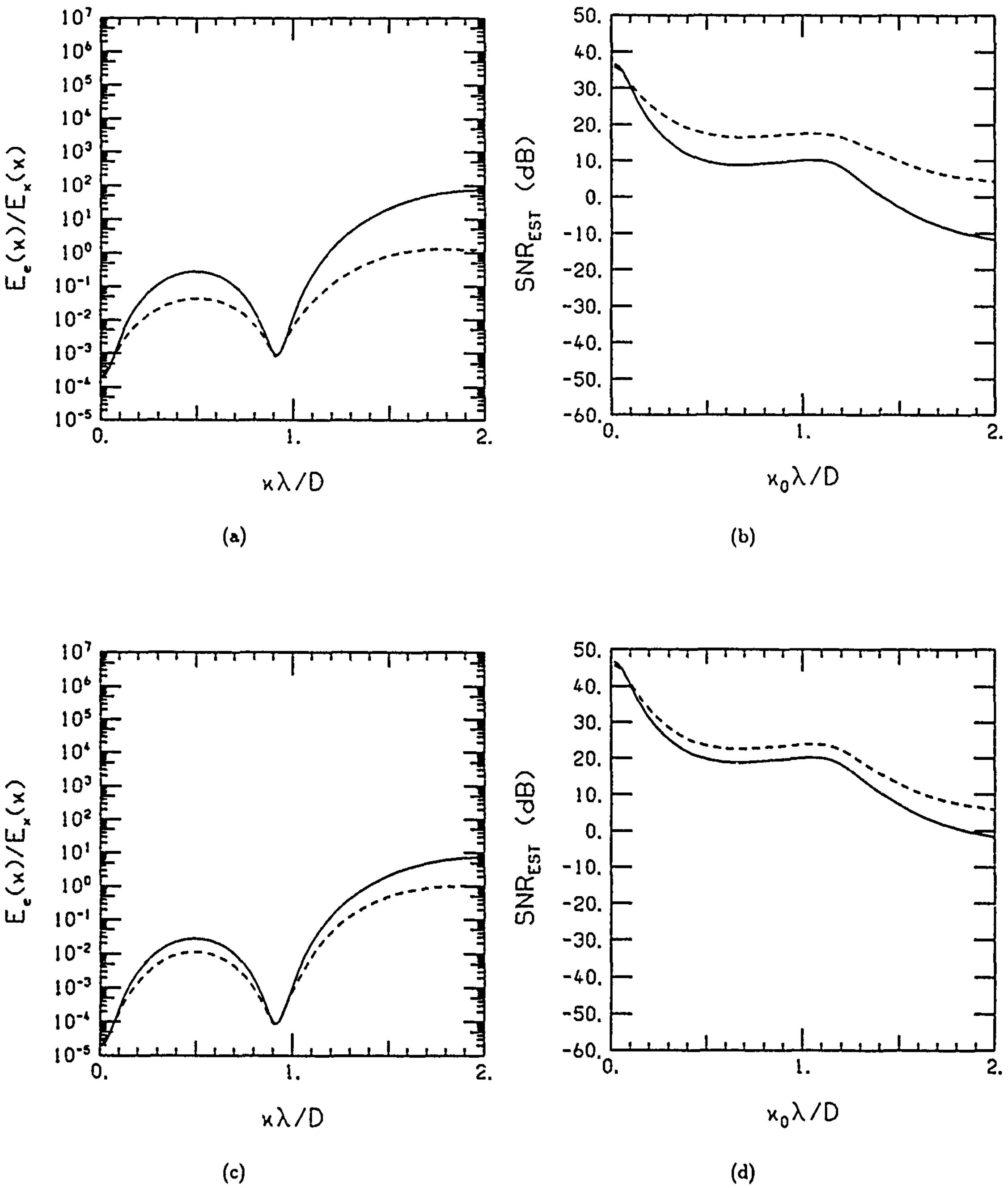


Figure 3.29. Energy spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number one and $d/D = 0.1, k = 63$ (corresponding to an image line length of 254 pixels or $63.5\lambda/D$), and an object support of 2 λ/D. Figures (a) and (b) correspond to SNR_{REF}=60 dB, and figures (c) and (d) correspond to SNR_{REF}=70 dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.

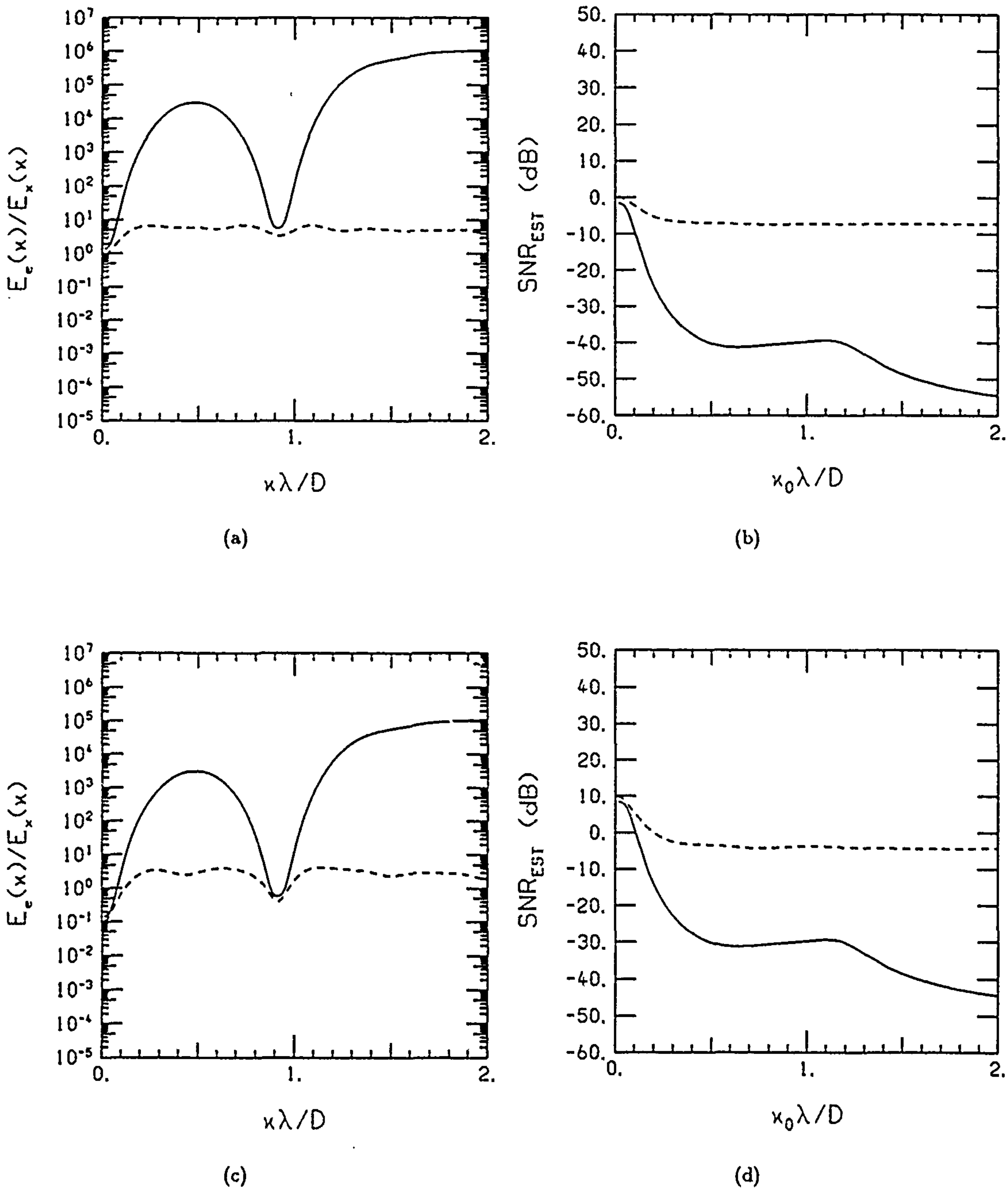


Figure 3.30. Energy spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number one and $d/D = 0.1, k = 63$ (corresponding to an image line length of 254 pixels or $63.5\lambda/D$), and an object support of 4 λ/D. Figures (a) and (b) correspond to SNR_{REF}=20 dB, and figures (c) and (d) correspond to SNR_{REF}=30 dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.

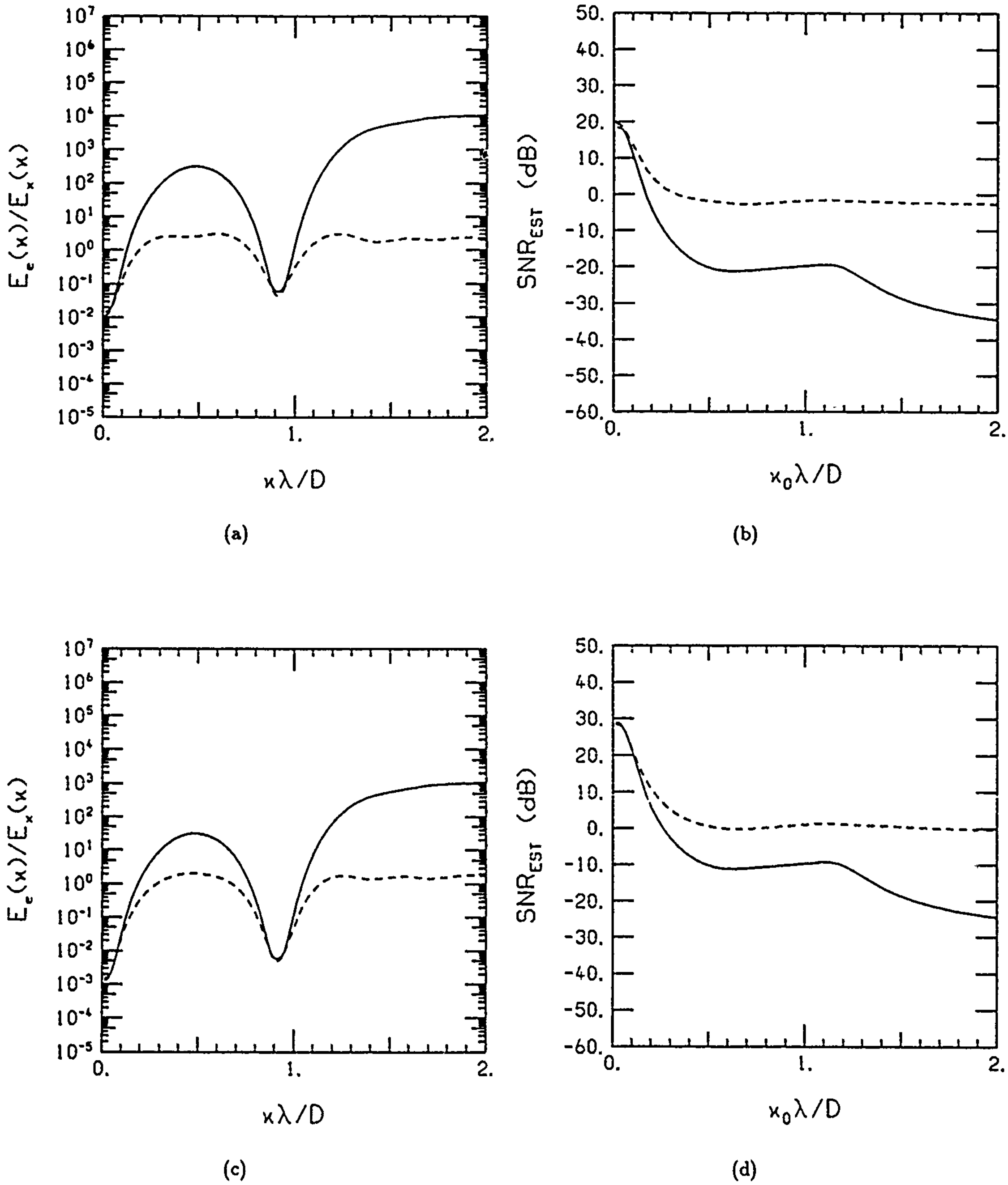


Figure 3.31. Energy spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number one and $d/D = 0.1, k = 63$ (corresponding to an image line length of 254 pixels or $63.5\lambda/D$), and an object support of 4 λ/D. Figures (a) and (b) correspond to SNR_{REF}=40 dB, and figures (c) and (d) correspond to SNR_{REF}=50 dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.

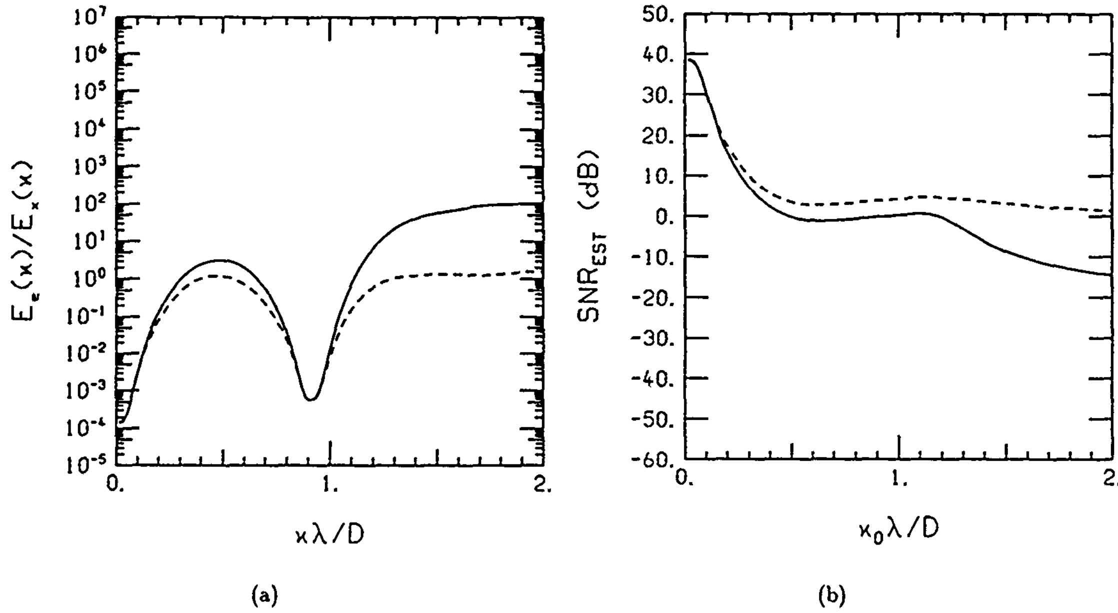


Figure 3.32. Energy Spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number one and $d/D = 0.1, k = 63$ (corresponding to an image line length of 254 pixels or $63.5\lambda/D$), an object support of 4 λ/D, and SNR_{REF}=60 dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.

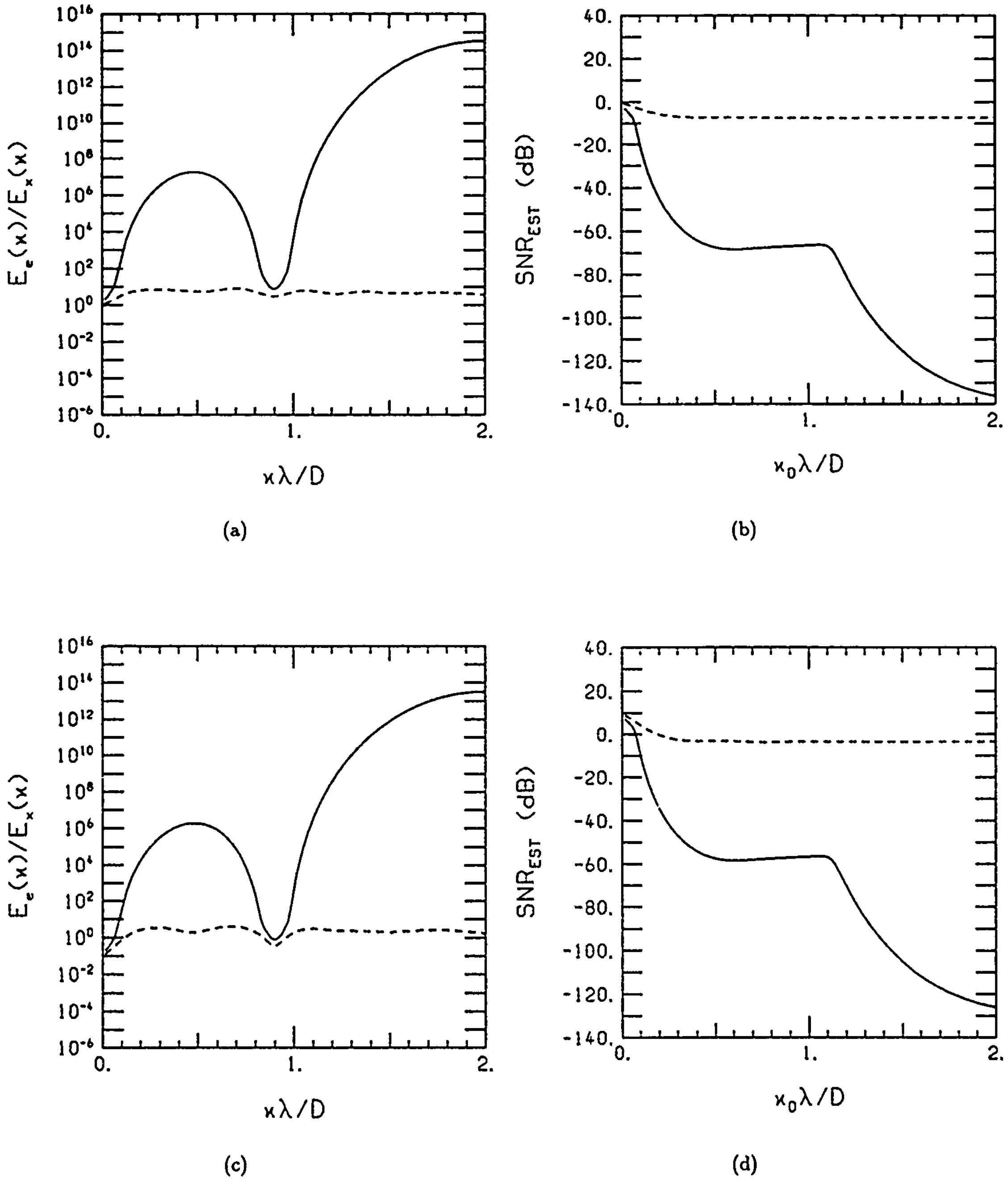


Figure 3.33. Energy spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number five and $d/D = 0.1, k = 511$ (corresponding to an image line length of 1150 pixels or $287.5\lambda/D$), and an object support of 4 λ/D. Figures (a) and (b) correspond to SNR_{REF}=20 dB, and figures (c) and (d) correspond to SNR_{REF}=30 dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.

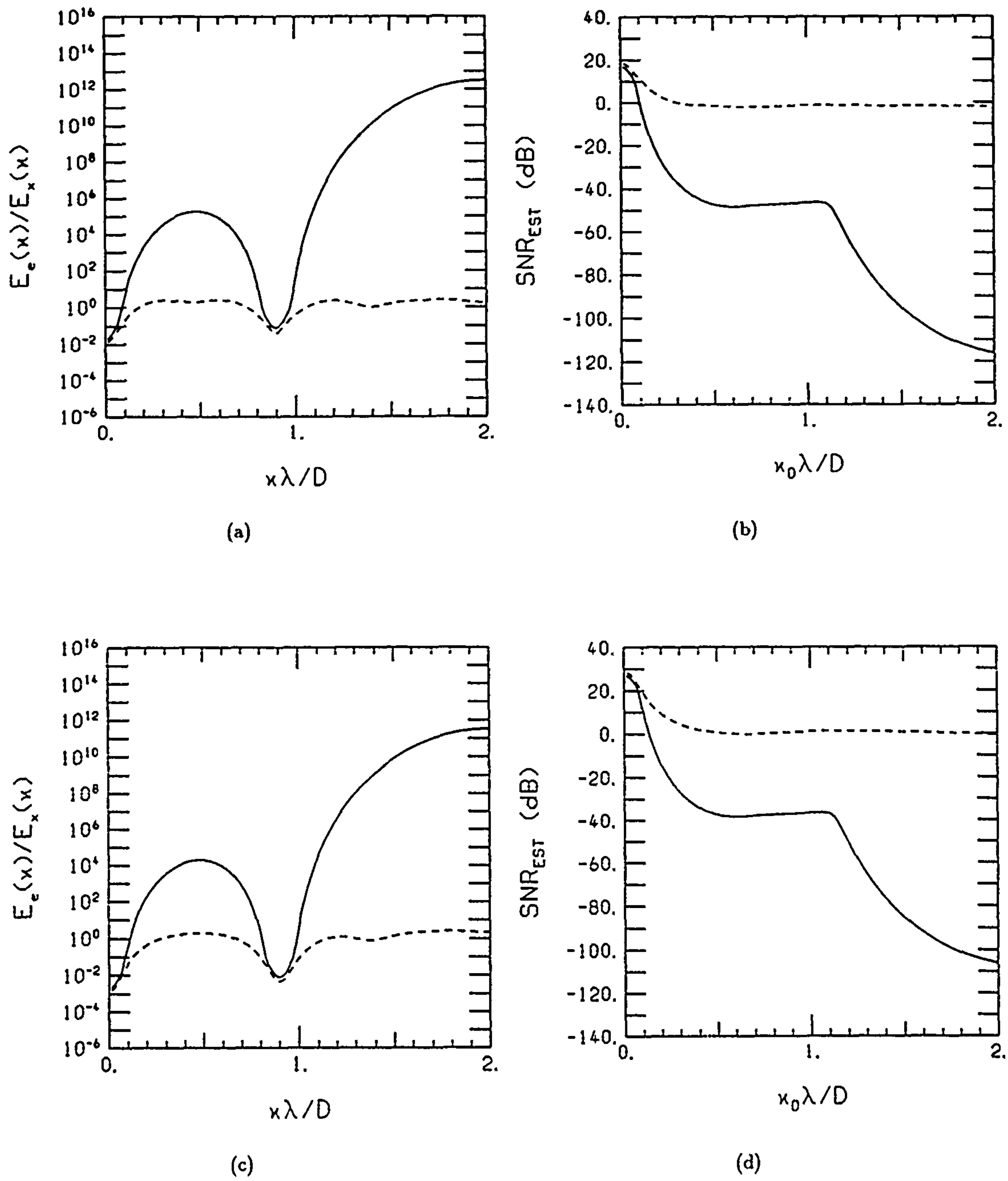


Figure 3.34. Energy spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number five and $d/D = 0.1, k = 511$ (corresponding to an image line length of 1150 pixels or $287.5\lambda/D$), and an object support of 4 λ/D. Figures (a) and (b) correspond to SNR_{REF}=40 dB, and figures (c) and (d) correspond to SNR_{REF}=50 dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.

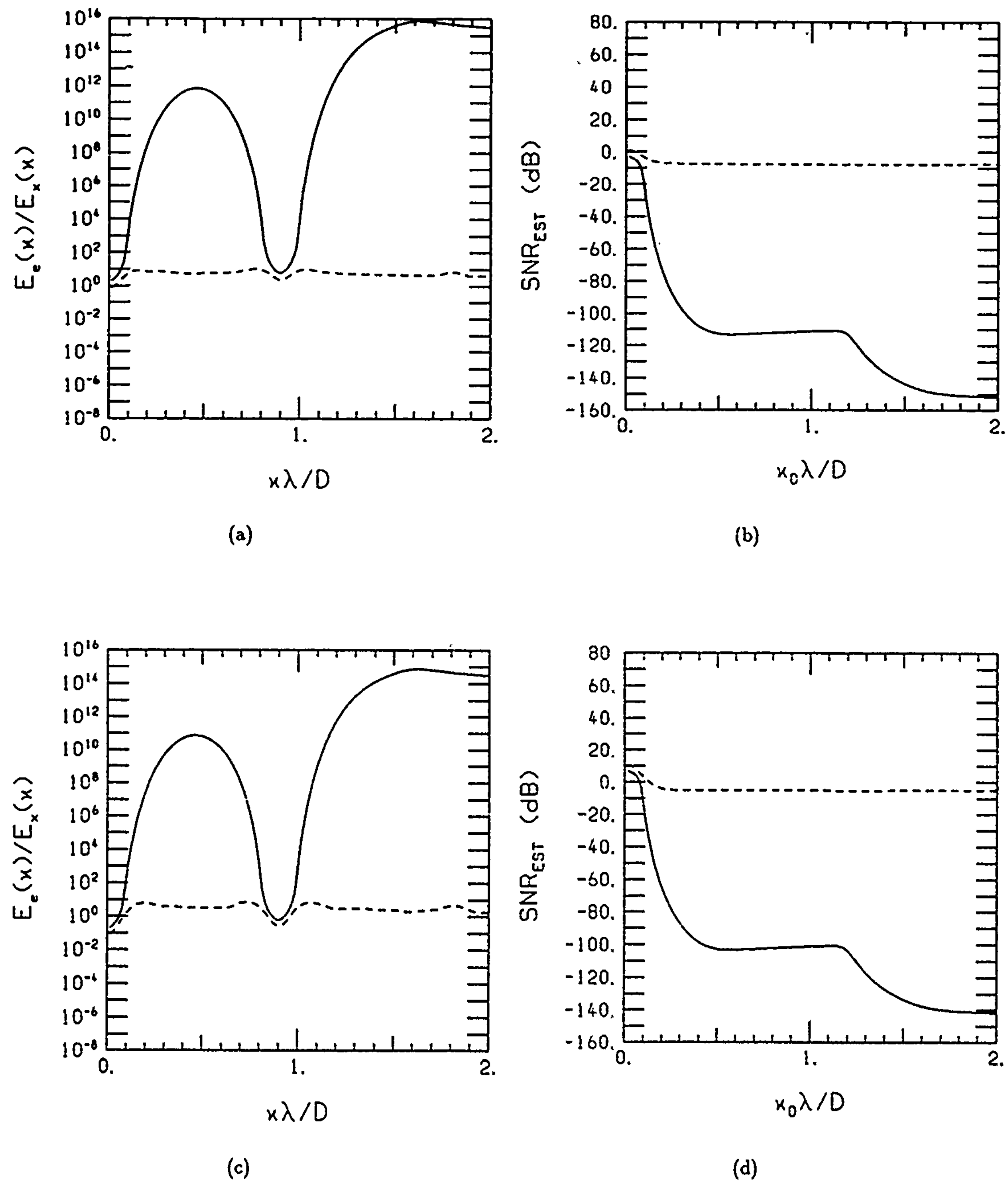


Figure 3.35. Energy spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number three and $d/D = 0.1, k = 255$ (corresponding to an image line length of 638 pixels or $159.5\lambda/D$), and an object support of 8 λ/D. Figures (a) and (b) correspond to SNR_{REF}=20 dB, and figures (c) and (d) correspond to SNR_{REF}=30 dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.

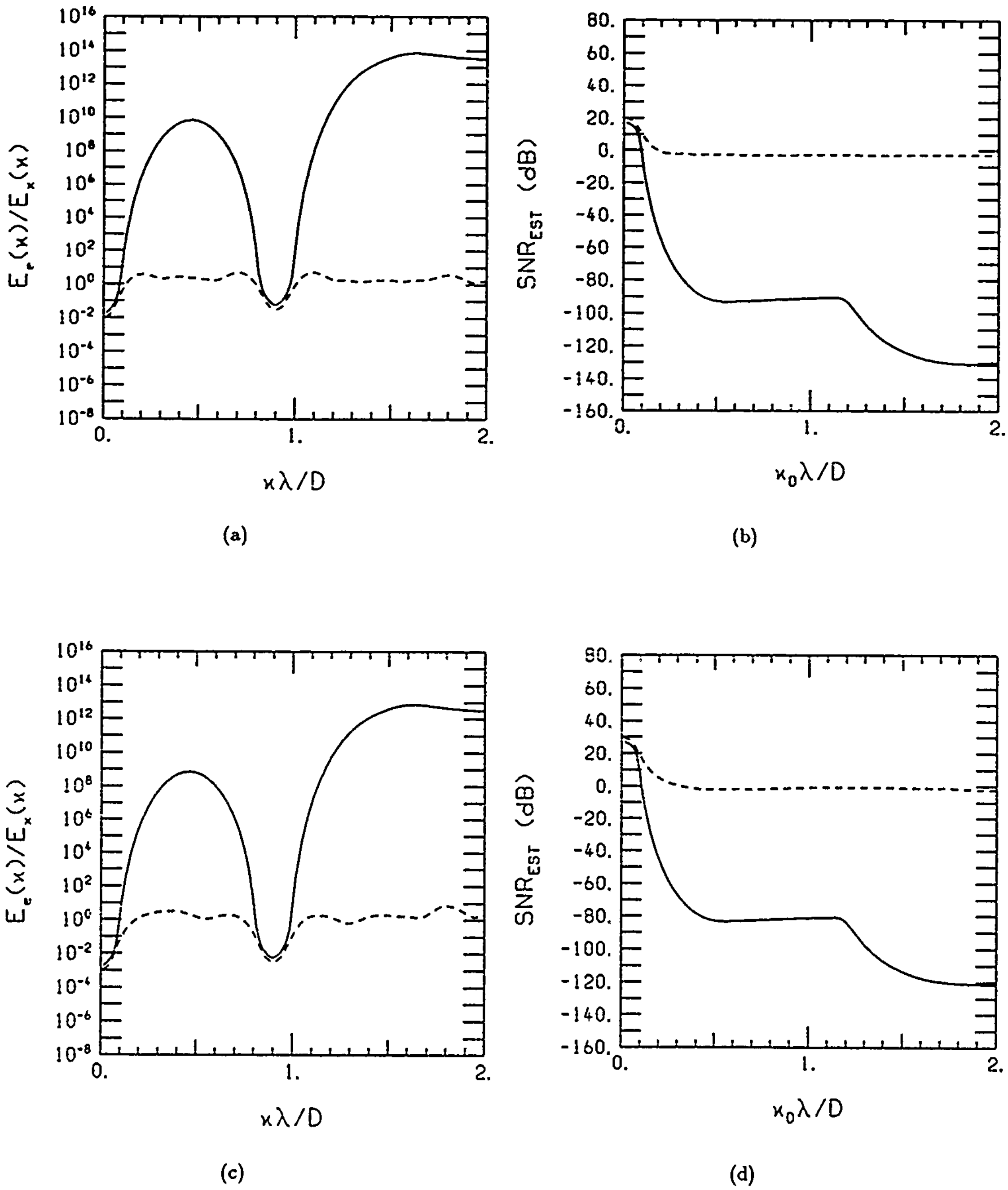


Figure 3.36. Energy spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number three and $d/D = 0.1, k = 255$ (corresponding to an image line length of 638 pixels or $159.5\lambda/D$), and an object support of 8 λ/D. Figures (a) and (b) correspond to SNR_{REF}=40 dB, and figures (c) and (d) correspond to SNR_{REF}=50 dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.

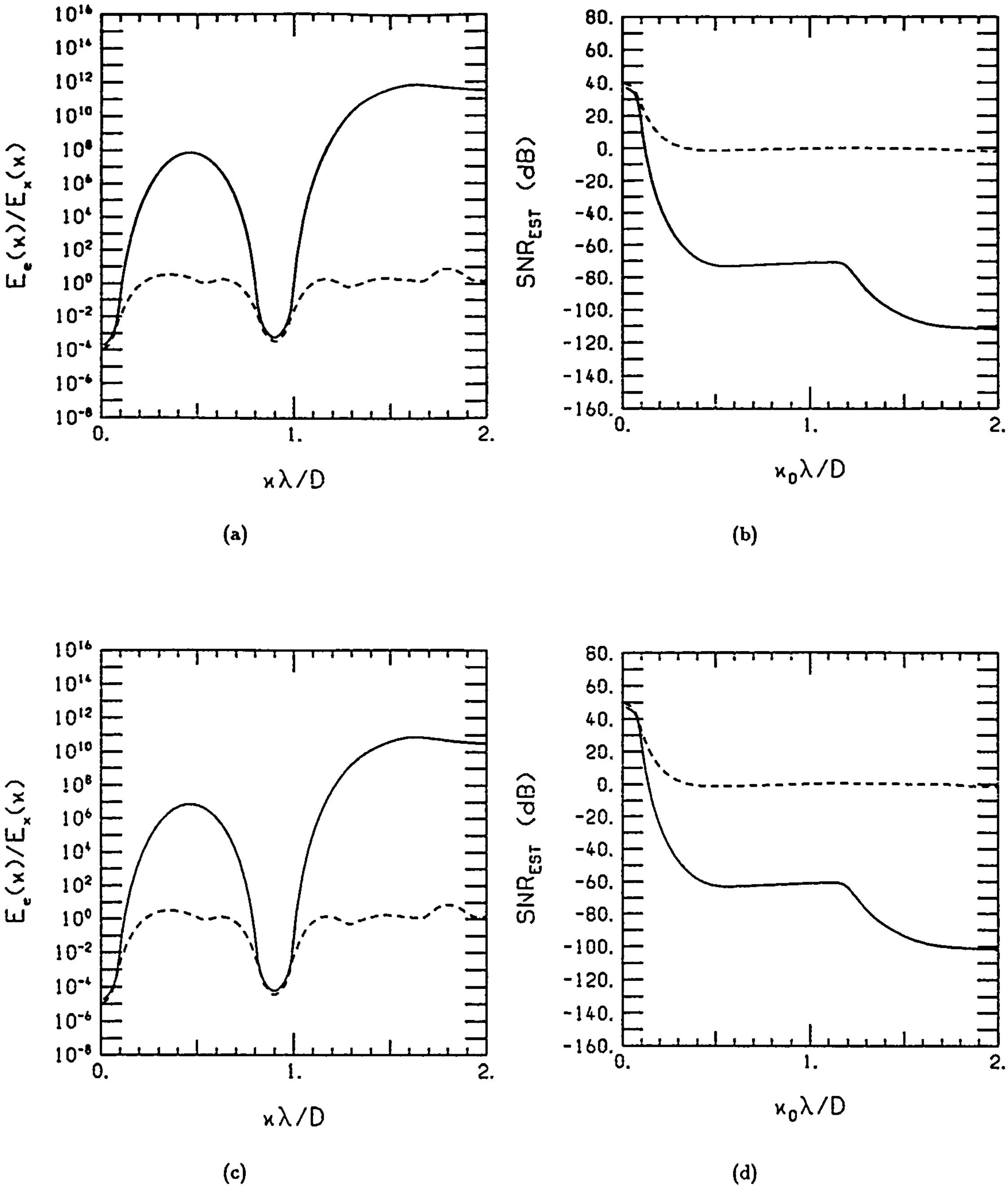


Figure 3.37. Energy spectra ratios and $\mathrm{SNR_{EST}}$ using the PSF of Eq. (3.1) with window number three and $d/D = 0.1, k = 255$ (corresponding to an image line length of 638 pixels or $159.5\lambda/D$), and an object support of 8 λ/D. Figures (a) and (b) correspond to $\mathrm{SNR_{REF}}$=60 dB, and figures (c) and (d) correspond to $\mathrm{SNR_{REF}}$=70 dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.

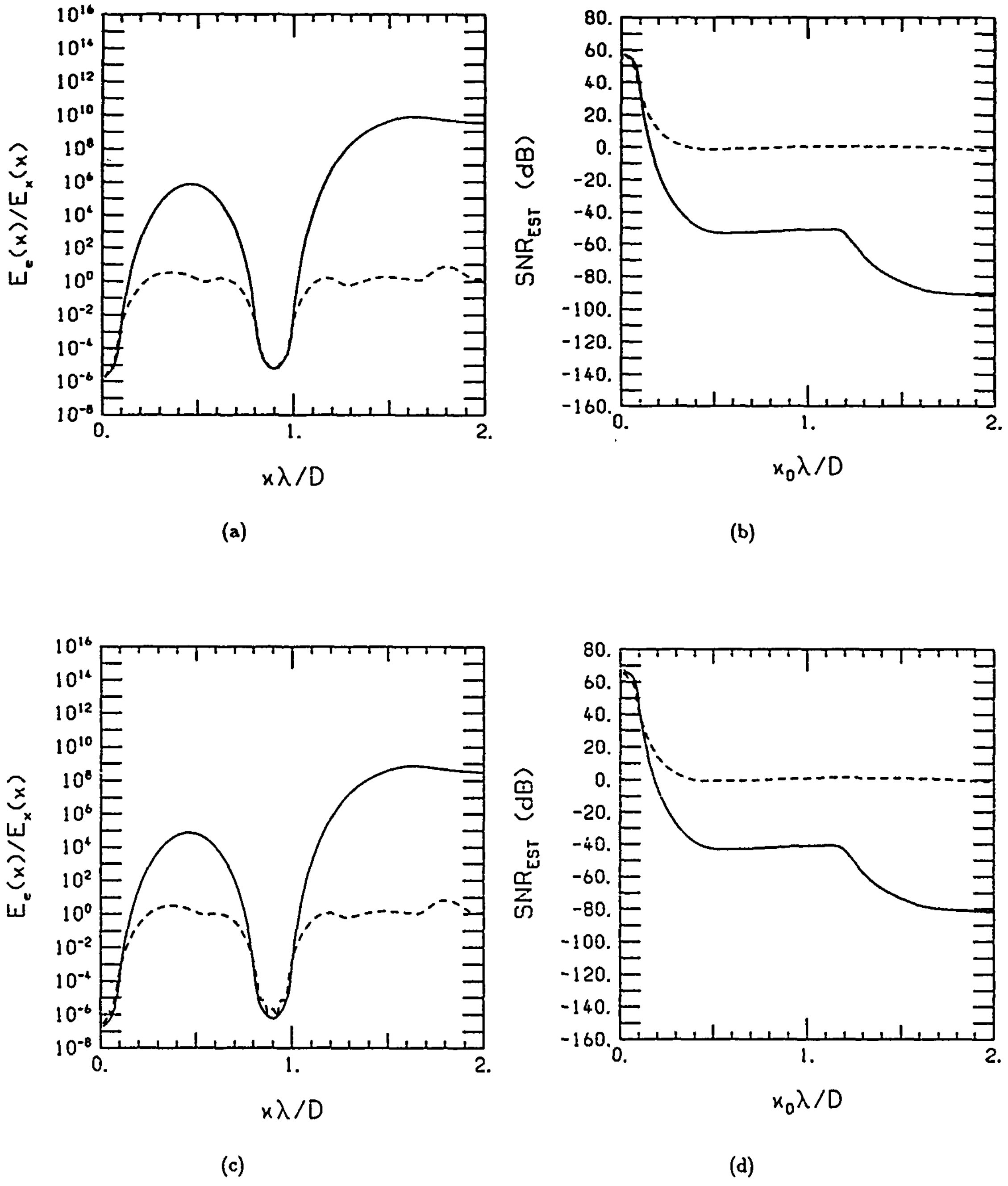


Figure 3.38. Energy spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number three and $d/D = 0.1, k = 255$ (corresponding to an image line length of 638 pixels or $159.5\lambda/D$), and an object support of 8 λ/D. Figures (a) and (b) correspond to SNR_{REF}=80 dB, and figures (c) and (d) correspond to SNR_{REF}=90 dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.

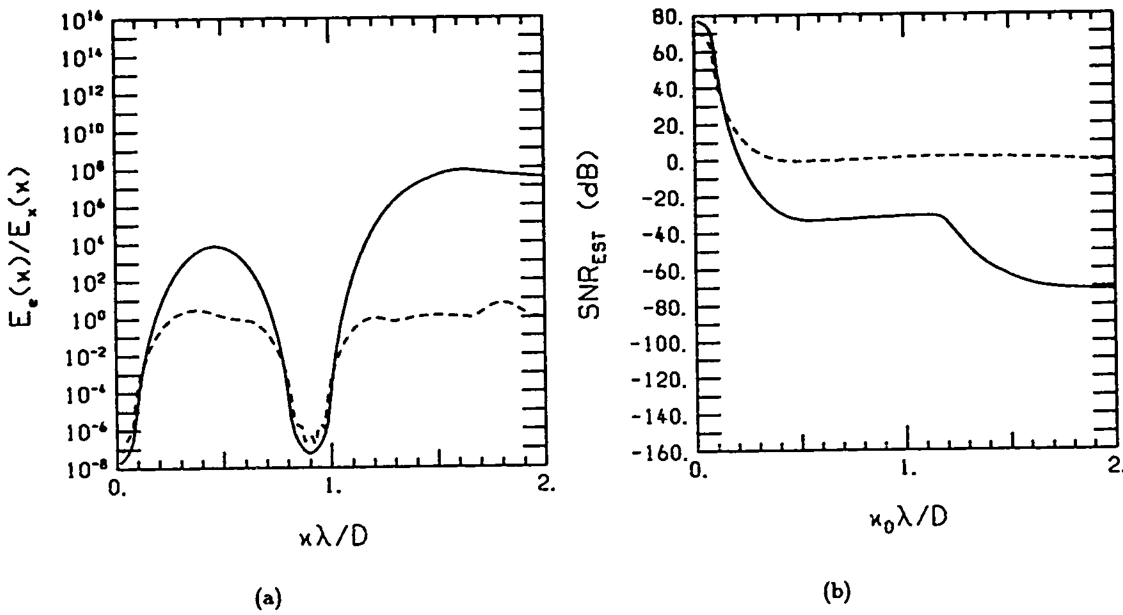


Figure 3.39. Energy Spectra ratios and SNR_{EST} using the PSF of Eq. (3.1) with window number three and $d/D = 0.1, k = 255$ (corresponding to an image line length of 638 pixels or $159.5\lambda/D$), an object support of 8 λ/D, and SNR_{REF}=100 dB. The solid curve in each figure corresponds to the least-squares solution with a finite support constraint only and the dashed curves correspond to the least-squares solution with both finite support and positivity constraints.

Chapter 4

Object Reconstruction with Sparse Arrays of Optical Apertures. Part I: Linear Methods

(originally issued as TR-1070)

4.1. Introduction

In this report we investigate the feasibility of recovering useful images of objects from noise-corrupted optical images formed by a very sparse array of apertures. By "useful image" we mean an image with a resolution commensurate with the overall dimension of the array and with a noise content low enough that the image would be considered useful. By "very sparse array" we mean an array of apertures with aperture size very small relative to aperture spacing so that the frequency response of the array comprises small "islands" of nonzero response surrounded by a "sea" of zero response.

At first, we questioned the feasibility of such a task based upon the supposition that superresolution is a dead issue, where here we are using the term "superresolution" rather loosely to mean filling in missing spatial frequency information with the use of some prior knowledge of the object. Had it not been shown that modest gains in resolution require enormous sacrifice in a signal-to-noise (SNR) ratio? However, it was brought to our attention that radio astronomers form apparently useful images from interferometric data using very sparse arrays and an algorithm called CLEAN.[2,27] Does CLEAN have magical powers, do radio astronomers have gobs of SNR at their disposal, or is there something about superresolution that we don't understand?

The possibility of resolving power beyond the classical limit of an idealized optical system has long been recognized.[19,22,18,20,21] The earliest mention of the subject appears to be by Coleman[19] in 1947. In 1952, Toraldo di Francia[22] applies the concept of super-gain in antennas to optical systems and concludes that the classical limit of 1.22 λ/D is only a practical limit and the actual resolution is limited only by noise. He discussed a procedure for designing what he calls a "super-resolving pupil" in which improved resolution can be obtained over a limited field by modifying the pupil of a diffraction-limited imaging system. In 1955, the same author[18] approached the concept of resolving power from the point of view of information theory. He reasoned that information is lost when an object is transformed into an image and therefore several different objects may produce the same image. If two different objects produce identical images then they cannot be "resolved", and thus this object ambiguity must have something to do with the definition of resolution. He gives examples for the coherent light case. He suggests that prior knowledge could be used to reduce the ambiguity in the object-image mapping. J. Harris[9] (1964) removes this difficulty by showing that no two distinctly different objects of finite angular size can have identical images. To establish this result, he uses two theorems from analytic function theory. The first theorem states that the Fourier transform of a square-integrable function of finite support is analytic throughout the entire domain of the spatial frequency plane. Harris then invokes analytic continuation (the second theorem) to demonstrate that, in the absence of noise, and starting with an arbitrarily small (but finite) piece of the Fourier transform of an object, one can find the entire Fourier transform (and thus the object itself). He next uses sampling theory to develop an algorithm to extrapolate from a piece of the Fourier transform of an object to the whole transform. The method requires solving a system of linear equations. Unfortunately, infinite precision requires an infinite number of equations. He applies his method to measuring the angular separation of two hypothetical point sources when the angle is less than the reciprocal of the spatial bandwidth of the system.

Following Harris, several researchers exploit the concept of analytic continuation to develop algorithms for object recovery. Barnes[7] (1966) uses prolate spheroidal wave functions to reconstruct objects of finite support in a one-dimensional coherent imaging system. Frieden[8] (1967) extends the use of prolate spheroidal wave functions to the reconstruction of partially coherently or incoherently illuminated 2-D objects of finite support. Both the methods of Barnes and Frieden require infinite series expansions in order to achieve infinite resolution. Brown[11] (1969) carries the development to the logical next step by examining the effect of series truncation on the amount of super-resolution achieved.

It was recognized early on that noise gums up the superresolution works. Rushforth and R. Harris [10] (1968) published one of the earliest papers in optics to seriously examine the effects of noise on superresolution performance. They looked at three types of noise in the context of a one-dimensional

coherent system and finite object support: background, measurement, and computer round off. Using the object recovery method of Barnes[7] and an extension of the method using Wiener filter theory, they computed mean-square-error as a function of the degree of extension of resolution beyond the classical limit with object support as a parameter. The paper demonstrates that even a modest extension of resolution beyond the classical limit is very costly in terms of the noisiness of the final image.

Lukosz[5,6] (1966, 1967) approach the subject of superresolution from a somewhat more global viewpoint. He proposed an invariance theorem to explain the concepts underlying all super-resolution techniques. The theorem states that it is not the spatial bandwidth of a system that is fixed, but the number of degrees of freedom, specified by the two space-bandwidth products and the time-bandwidth product of the optical system. Thus, one can extend any parameter of the system, e.g., spatial bandwidth, beyond the classical limit by proportionately reducing some other parameter, provided some *a priori* information concerning the object, e.g., independence of time, is known.

A few researchers approached object restoration from an information-theoretical standpoint.[17,18] As early as 1955, Fellgett and Linfoot[23] derived the information capacity of a two-dimensional optical system in terms of SNR, field-of-view, and spatial bandwidth. A recent paper by Cox and Sheppard[1] (1987) extended the results of Fellgett and Linfoot by including exposure time and temporal bandwidth to the expression for the information capacity of an optical system. They then use the invariance of capacity to demonstrate that any attempt to increase the spatial bandwidth of the optical system through analytic continuation results in a reduction of the SNR in the final image. They derive an upper bound on the resolution improvement as a function of the ratio of SNR with and without the resolution improvement. The bound is very loose, but nonetheless demonstrates that a modest improvement in resolution can require very large increases in signal strength.

In the research described so far, the concept of superresolution was generally thought of as any process which extends spatial frequency knowledge of an object beyond the spatial cutoff frequency of the optical system. In a series of three papers, Lannes et.al.[24,25,26] (1987) analyze the problem of object restoration with missing spectral information. They view the restoration problem to be a compromise between resolution and robustness. They develop a robustness theory that includes object support and the distribution of regions of missing spectral information and conclude that super-resolution extrapolation is "harder" than super-resolution interpolation. In other words, extrapolating frequency information is "hard", but interpolating between known regions, such as CLEAN does, may be "easier". Furthermore, they state that the robustness of the interpolation process is increased whenever the frequency gaps are well distributed over the aperture to be synthesized.

To simplify computations and interpretation of results, we decided to conduct our investigation in one dimension. Furthermore, we chose a discrete model of the optical system so that convolution could be simply represented as matrix multiplication. A "typical" sparse array in one dimension is shown in Fig. 4.1. This particular array is an example of a nonredundant array, so called because no pair of subapertures contributes the same spatial frequency region to the MTF as any other pair (except the region in the vicinity of zero, of course). This is witnessed by noting that all "islands" of nonzero response are of equal height (again, except for the "island" at the origin). In our studies, we will be particularly interested in the feasibility of using the concept of finite support to "fill in" the gaps shown in Fig. 4.1, when the subaperture size (d) is very small compared to the array size (D), or, put another way, when the width of the "islands" in the MTF is small compared to the spacing between islands. The results we will present were computed using nonredundant arrays of four, five and six subapertures, shown respectively in Fig.'s 4.2, 4.3, and 4.4.* It is in no way important to the results we present that the arrays are nonredundant. It is merely a convenience. An important feature to note of each of the arrays is the gap between islands. Our results will show that when

* These arrays were obtained from Barakat.[4]

the angular support of the object is less than the inverse of this gap, that the price, in terms of signal-to-noise ratio, that we have to pay for using a sparse array is "moderate", and otherwise, the price rapidly becomes "severe".

We begin in Section 4.2 with a description of the mathematical model of the image formed by our one-dimensional discrete optical system. In Section 4.3 we define a performance measure. It remains to pick an object recovery algorithm. There are probably at least as many recovery algorithms as there are researchers in the filed.[3,12,13,14,15,16] Most of the algorithms are iterative, mainly because of the practical limitations of the object recovery process relative to the requirements of the two dimensional problem, but also because of the ease of incorporating prior knowledge such as positivity. Since we have no desire to develop yet another algorithm, but rather are interested in "feasibility" in terms of signal-to-noise ratio, we will go on the assumption that a linear transformation of the "measurement" will probably yield performance adequate for our purposes. We thus choose two linear object recovery methods, minimum-variance, where knowledge of the first two moments of object intensity distribution and observation noise is assumed, and unweighted least-squares, where no statistical knowledge is used. The minimum-variance results are presented in Section 4.4 and those of least-squares in Section 4.5.

4.2. Discrete Optical Model (One-Dimensional)

Let the components of the $L \times 1$ vector $\mathbf{z}$ be the intensity pixels of the object line, let the components of the $N \times 1$ vector $\mathbf{y}$ represent the intensity pixels of the image line, and let the $N \times L$ matrix B be the transformation from object line to image line (its columns contain shifted versions of the system point-spread-function). Then an image can be represented by the matrix equation

$$\mathbf{y} = B\mathbf{z} + \mathbf{n}, \tag{4.1}$$

where $\mathbf{n}$ is a $N \times 1$ noise vector. The product $B\mathbf{z}$ in Eq. (4.1) represents convolution of the object line with the point spread-function, followed by truncation of the image. Let the point spread function be $h(k)$. Then the B matrix is

$$B = \begin{bmatrix}
h(-K) & h(-K-1) & \dots & h(-K-L+1) \\
h(-K+1) & h(-K) & \dots & h(-K-L+2) \\
\vdots & h(-K+1) & \dots & \vdots \\
\vdots & \vdots & \ddots & \vdots \\
\vdots & \vdots & \dots & h(-K-2) \\
h(0) & \vdots & \dots & h(-K-1) \\
\vdots & h(0) & \dots & h(-K) \\
\vdots & \vdots & \dots & h(-K+1) \\
h(K-1) & \vdots & \dots & \vdots \\
h(K) & h(K-1) & \dots & \vdots \\
h(K+1) & h(K) & \dots & h(0) \\
h(K+2) & h(K+1) & \dots & \vdots \\
\vdots & h(K+2) & \dots & \vdots \\
\vdots & \vdots & \dots & h(K-1) \\
h(K+L-1) & h(K+L-2) & \dots & h(K)
\end{bmatrix}, \tag{4.2}$$

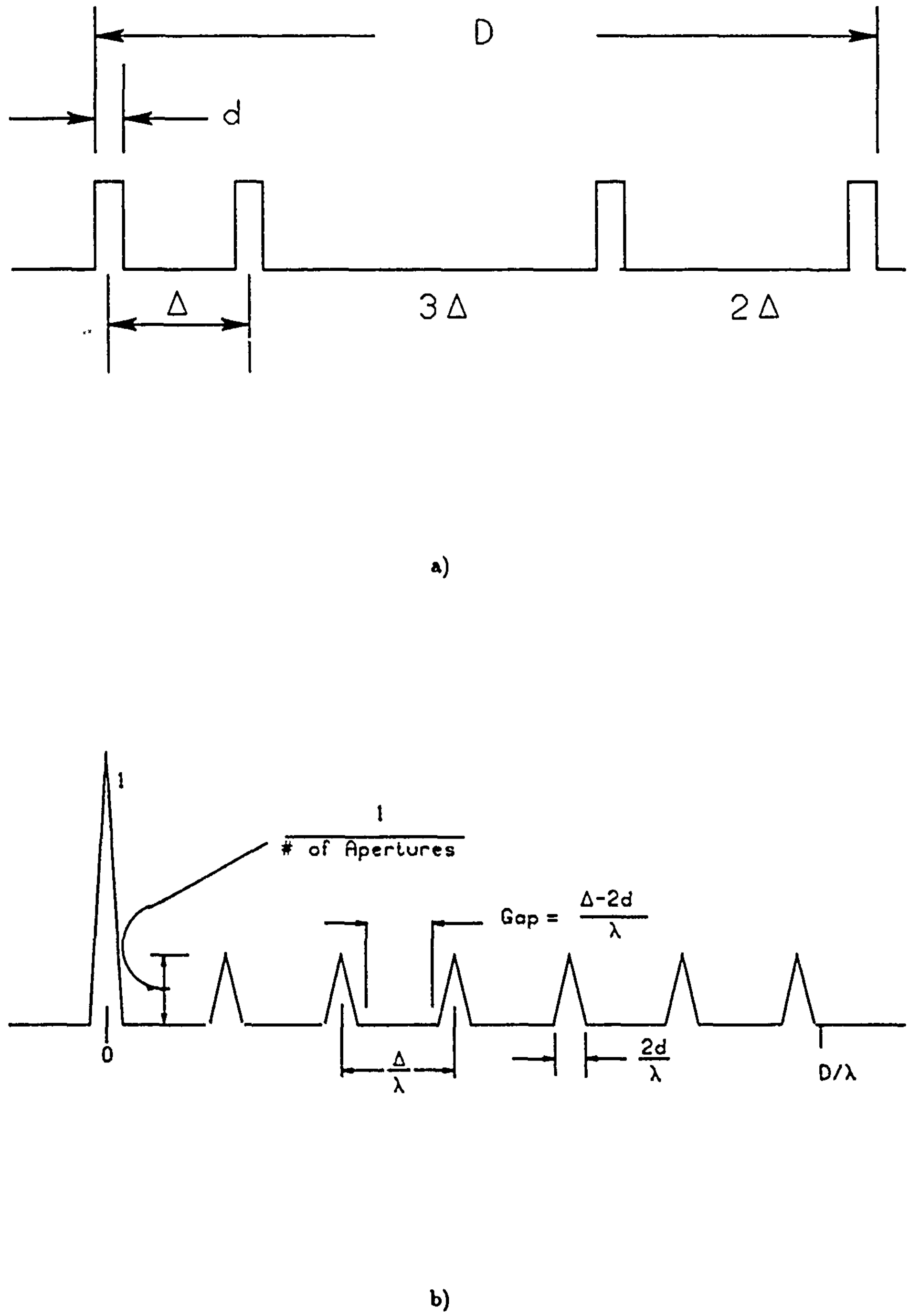


Figure 4.1.
Typical nonredundant sparse array aperture function (a) and the corresponding MTF (b).

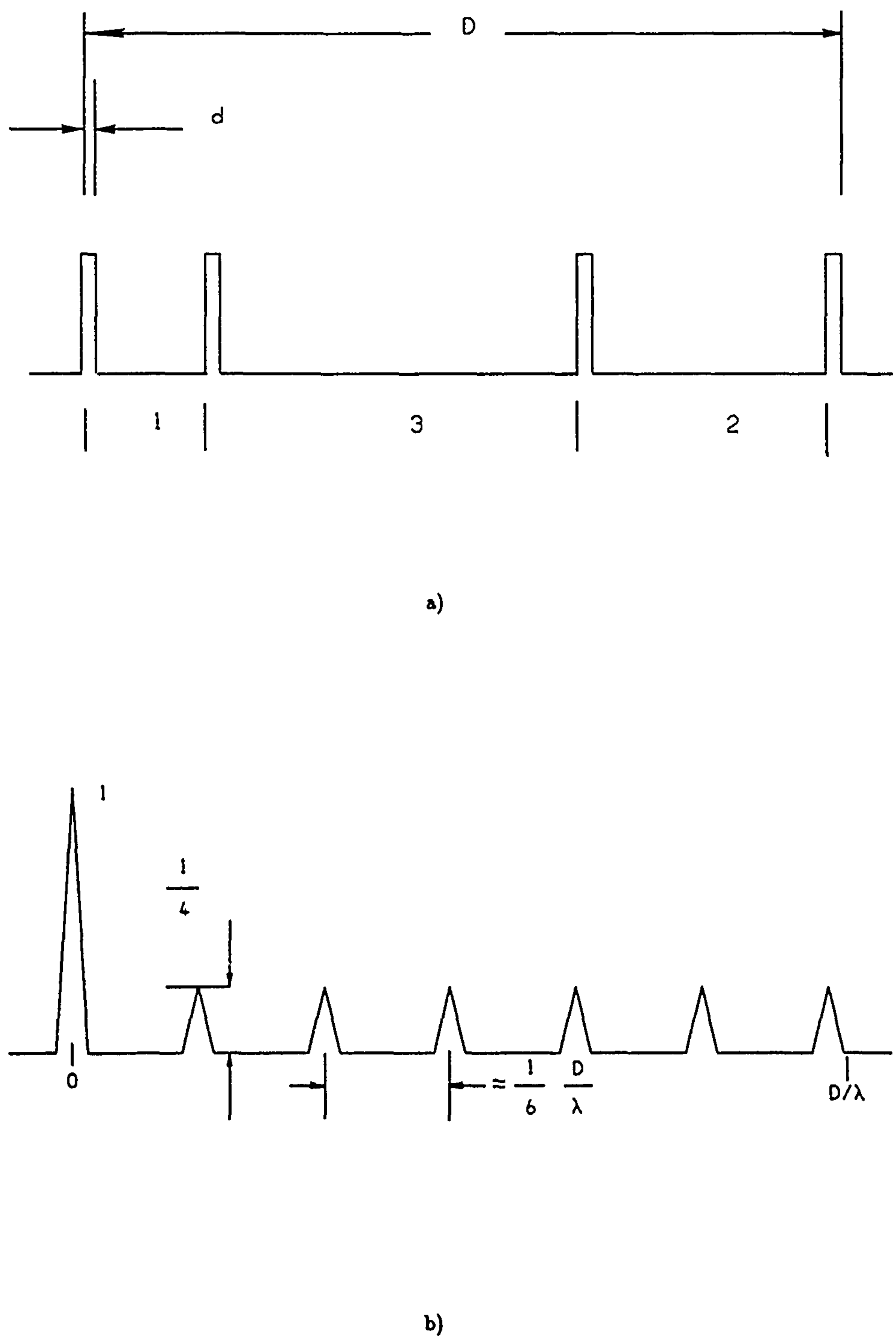
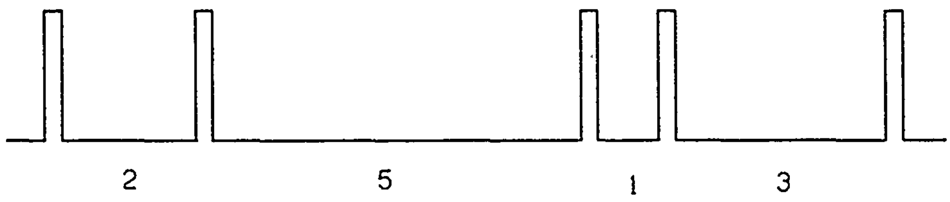
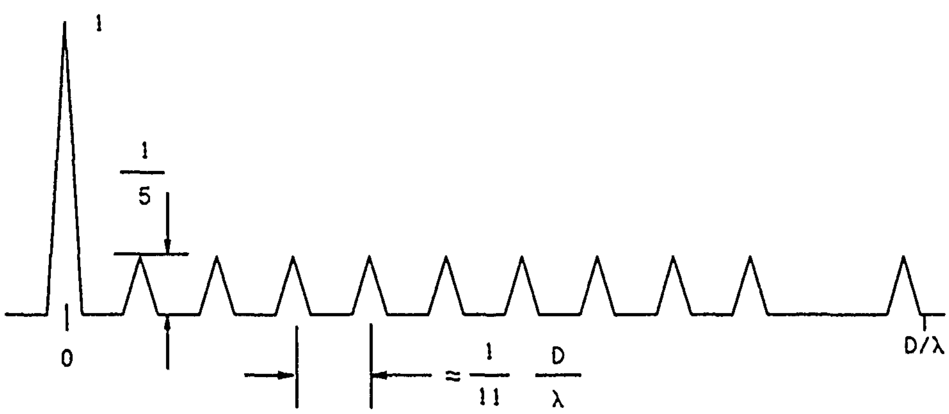


Figure 4.2.
Sparse array aperture function (a) and the corresponding MTF (b) for a nonredundant four-subaperture array.



a)



b)

Figure 4.3.
Sparse array aperture function (a) and the corresponding MTF (b) for a nonredundant five-subaperture array.

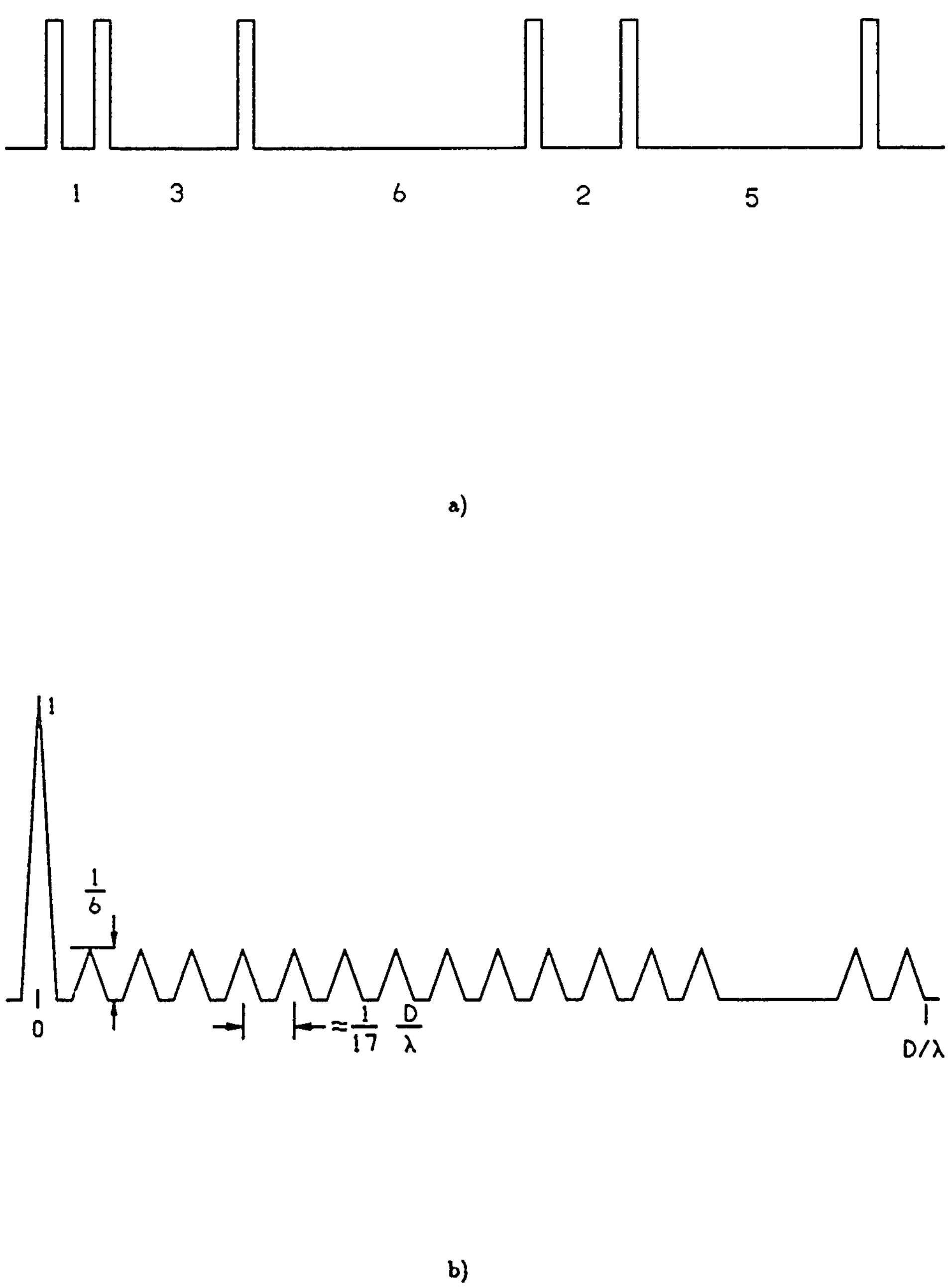


Figure 4.4.
Sparse array aperture function (a) and the corresponding MTF (b) for a nonredundant six-subaperture array.

where $K = (N - L)/2$.

Now let the $M \times 1$ vector $\mathbf{x}$ represent an object of finite contagious support of length $M \leq L$ pixels located somewhere in the object line. We can write

$$\mathbf{z} = W\mathbf{x}, \tag{4.3}$$

where the $L \times M$ matrix W is of the form

$$W = \begin{bmatrix} & & 0 & \\ \hline 1 & & & 0 \\ & 1 & & \\ & & \ddots & \\ 0 & & & 1 \\ \hline & & 0 & \end{bmatrix}. \tag{4.4}$$

Thus, W comprises an $M \times M$ identity matrix embedded in a $L \times M$ matrix of zeros. Combining Eq.'s (4.3) and (4.1) yields

$$\mathbf{y} = BW\mathbf{x} + \mathbf{n}. \tag{4.5}$$

To complete the model, we need a point-spread function. Let $w(x)$ be an arbitrary aperture function. The corresponding modulation transfer function (MTF) is given below, where κ is the spatial frequency variable in units of cycles per radian and λ is wavelength. We write

$$MTF(\kappa) = \frac{1}{A}\int dx\, w(x + \tfrac{1}{2}\kappa\lambda)w(x - \tfrac{1}{2}\kappa\lambda) \tag{4.6}$$

A is chosen to be the area (in this case the length) of the aperture, so that the MTF at zero spatial frequency is unity.

The point-spread function of an optical system is the inverse Fourier transform of the MTF function, or

$$h_c(x) = \int d\kappa\, \mathrm{MTF}(\kappa)\exp[2\pi i\kappa x]. \tag{4.7}$$

The subscript c denotes that this point-spread function is the continuous version. The discrete version of the point-spread function is found by sampling and scaling the continuous version. Let δ denote the pixel spacing in the image line. Then the discrete point spread function used in the B matrix given by Eq. (4.2) is

$$h(n) = \delta h_c(n\delta); \quad n = 0, \pm 1, \pm 2, \cdots. \tag{4.8}$$

The continuous point spread functions for the four, five, and six subaperture arrays are given in Eq.'s (9), (10), and (11) respectively. (The reader should see Fig.'s 4.2, 4.3, and 4.4 for the corresponding MTF's.) We have

$$h_c(x) = \frac{d}{\lambda}\left(\frac{\sin \pi\frac{d}{\lambda}x}{\pi\frac{d}{\lambda}x}\right)^2 \left(1 + \tfrac{1}{2}\sum_{i=1}^{6} \cos\left[i\frac{\pi}{3}\frac{D}{\lambda}\left(1 - \frac{d}{D}\right)x\right]\right), \tag{4.9}$$

$$h_c(x) = \frac{d}{\lambda}\left(\frac{\sin \pi\frac{d}{\lambda}x}{\pi\frac{d}{\lambda}x}\right)^2 \left(1 + \frac{2}{5}\sum_{\substack{i=1\\ i\neq 10}}^{11} \cos\left[i\frac{2\pi}{11}\frac{D}{\lambda}\left(1 - \frac{d}{D}\right)x\right]\right), \tag{4.10}$$

$$h_c(x) = \frac{d}{\lambda}\left(\frac{\sin \pi\frac{d}{\lambda}x}{\pi\frac{d}{\lambda}x}\right)^2 \left(1 + \tfrac{1}{3}\sum_{\substack{i=1\\ i\neq 14\\ i\neq 15}}^{17} \cos\left[i\frac{2\pi}{17}\frac{D}{\lambda}\left(1 - \frac{d}{D}\right)x\right]\right). \tag{4.11}$$

4.3 A Performance Measure

In the sections to follow we investigate two algorithms for object recovery. In order to evaluate the performance of each algorithm and to compare algorithms, we need a measure of performance that we can apply uniformly to both algorithms. Our object recovery algorithms not only attempt to fill in the gaps between measured spatial frequency components of the object (interpolative super-resolution), but also to estimate the object at spatial frequencies beyond D/λ (extrapolative super-resolution), where D is the over all length of the array. Because of array geometries, our algorithms will be performing better at recovering some spatial frequency components than others. We therefore need a frequency sensitive performance measure. To this end, let $x(n)$ be the n^{th} pixel of the object, $\hat{x}(n)$ be the n^{th} pixel of an estimate of the object, and $e(n)$ be the n^{th} pixel of the estimation error given by

$$e(n) = x(n) - \hat{x}(n), \tag{4.12a}$$

or in vector notation,

$$\mathbf{e} = \mathbf{x} - \hat{\mathbf{x}}. \tag{4.12b}$$

One common measure of performance is mean-square-error, $\langle \mathbf{e}^T \mathbf{e} \rangle$. For our purposes this measure is not useful. It gives us no indication of how well an algorithm performs as a function of spatial frequency. Instead, the performance measure we will use is a signal-to-noise ratio (SNR) that is a function of a spatial cutoff frequency. The behavior of the SNR as a function of cutoff frequency will be an indication of the spatial resolution of the sparse array. To introduce frequency sensitivity into the performance measure, let $x_{\kappa_0}(n)$, $\hat{x}_{\kappa_0}(n)$, and $e_{\kappa_o}(n)$ be filtered versions of $x(n)$, $\hat{x}(n)$, and $e(n)$, respectively, where the filtering is ideal low-pass with cutoff frequency κ_o. To explicitly define the filtering operation, let $y(n)$ be a sequence of pixels and $\tilde{y}(\kappa)$ be its Fourier transform given by

$$\tilde{y}(\kappa) = \sum_n y(n) \exp(-2\pi i n \kappa \delta), \tag{4.13}$$

where δ is the object pixel spacing. Let $\tilde{f}(\kappa)$ be the filter transfer function given by

$$\tilde{f}(\kappa) = \begin{cases} 1, & |\kappa| \le \kappa_o \\ 0, & |\kappa| > \kappa_o. \end{cases} \tag{4.14}$$

Then the Fourier transform of $y_{\kappa_o}(n)$, the filtered version of $y(n)$, is given by

$$\tilde{y}_{\kappa_o}(\kappa) = \tilde{y}(\kappa)\tilde{f}(\kappa). \tag{4.15}$$

Thus, we have

$$\tilde{x}_{\kappa_o}(\kappa) = \tilde{x}(\kappa)\tilde{f}(\kappa) \tag{4.16}$$

$$\tilde{\hat{x}}_{\kappa_o}(\kappa) = \tilde{\hat{x}}(\kappa)\tilde{f}(\kappa) \tag{4.17}$$

$$\begin{aligned} \tilde{e}_{\kappa_o}(\kappa) &= \tilde{e}(\kappa)\tilde{f}(\kappa) \\ &= \left[\tilde{x}(\kappa) - \tilde{\hat{x}}(\kappa)\right] \tilde{f}(\kappa) \\ &= \tilde{x}_{\kappa_o}(\kappa) - \tilde{\hat{x}}_{\kappa_o}(\kappa). \end{aligned} \tag{4.18}$$

Let us now define a signal-to-noise ratio for the filtered estimate $\hat{x}_{\kappa_o}(n)$ as the ratio of the average energy of the filtered object to the average energy of the filtered estimation error:

$$\mathrm{SNR_{EST}}(\kappa_o) = \frac{\left\langle \sum_n x^2_{\kappa_o}(n) \right\rangle}{\left\langle \sum_n e^2_{\kappa_o}(n) \right\rangle}. \tag{4.19}$$

The angle brackets in Eq. (4.19) denote ensemble average. From Parseval's theorem we know that the quantities inside the angle brackets in Eq. (4.19) can be written in terms of their Fourier transforms, as follows:

$$\sum_n x_{\kappa_o}^2(n) = \delta \int_{-1/2\delta}^{1/2\delta} d\kappa \, |\widetilde{x}_{\kappa_o}(\kappa)|^2 \tag{4.20}$$

$$\sum_n e_{\kappa_o}^2(n) = \delta \int_{-1/2\delta}^{1/2\delta} d\kappa \, |\widetilde{e}_{\kappa_o}(\kappa)|^2. \tag{4.21}$$

We can use Eq.'s (4.13), (4.14), (4.16), and (4.18) to rewrite Eq.'s (4.20) and (4.21), yielding

$$\begin{aligned} \sum_n x_{\kappa_o}^2(n) &= \delta \int_{-1/2\delta}^{1/2\delta} d\kappa \, |\widetilde{x}(\kappa)|^2 |\widetilde{f}(\kappa)|^2 \\ &= 2\delta \int_0^{\kappa_o} d\kappa \, |\widetilde{x}(\kappa)|^2 \end{aligned} \tag{4.22}$$

$$\begin{aligned} \sum_n e_{\kappa_o}^2(n) &= \delta \int_{-1/2\delta}^{1/2\delta} d\kappa \, |\widetilde{e}(\kappa)|^2 |\widetilde{f}(\kappa)|^2 \\ &= 2\delta \int_0^{\kappa_o} d\kappa \, |\widetilde{e}(\kappa)|^2. \end{aligned} \tag{4.23}$$

Replacing the quantities inside the angle-brackets in Eq. (4.19) with their corresponding expressions given by Eq.'s (4.22) and (4.23), and bringing the angle-brackets inside the integrals, yields

$$\mathrm{SNR_{EST}}(\kappa_o) = \frac{\int_0^{\kappa_o} d\kappa \, \langle |\widetilde{x}\kappa|^2 \rangle}{\int_0^{\kappa_o} d\kappa \langle |\widetilde{e}(\kappa)|^2 \rangle}. \tag{4.24}$$

Since integrating the quantities $\delta\langle |\widetilde{x}(\kappa)|^2\rangle$ and $\delta\langle |\widetilde{e}(\kappa)|^2\rangle$ over the spatial frequency band $|\kappa| \leq \kappa_o$ yields, respectively, the energy in the object and estimation error over the same band, we will call these quantities energy spectral densities or energy spectra, and use the notation

$$E_x(\kappa) = \delta\langle |\widetilde{x}(\kappa)|^2 \rangle \tag{4.25}$$

$$E_e(\kappa) = \delta\langle |\widetilde{e}(\kappa)|^2 \rangle. \tag{4.26}$$

So that

$$\mathrm{SNR_{EST}}(\kappa_o) = \frac{\int_0^{\kappa_o} d\kappa \, E_x(\kappa)}{\int_0^{\kappa_o} d\kappa \, E_e(\kappa)}. \tag{4.27}$$

To compute the energy spectrum of a random sequence, we simply follow the prescription given by Eq.'s (4.25) and (4.13). Let $\mathbf{z}$ be a random vector with n^{th} component $z(n)$. Then its energy spectrum is given by

$$\begin{aligned} E_z(\kappa) &= \delta\langle |\widetilde{z}(\kappa)|^2 \rangle \\ &= \delta \left\langle \left| \sum_n z(n) \exp(-2\pi i n\kappa\delta) \right|^2 \right\rangle \\ &= \delta \sum_n \sum_m \langle z(n) z(m) \rangle \exp[-2\pi i (n-m)\kappa\delta]. \end{aligned} \tag{4.28}$$

We note from Eq. (4.28) that we need *all* entries of the covariance matrix of $z(n)$ in the evaluation of its energy spectrum.

4.4. Minimum-Variance Processor

The first processor (estimator) we will investigate is the so-called minimum variance processor. We assume that we have first and second moment information on the object vector $\mathbf{x}$ and the noise vector $\mathbf{n}$ of Eq. (4.5), i.e., we know the mean vector and covariance matrices of $\mathbf{x}$ and $\mathbf{n}$. Since we know the mean values of $\mathbf{x}$ and $\mathbf{n}$, and we can compute the mean value of $\mathbf{y}$, we will assume that the mean values have been subtracted out of Eq. (4.5), and our estimate $\widehat{\mathbf{x}}$ is the deviation of $\mathbf{x}$ from its mean value. In other words, we have the observation model

$$\mathbf{y} = BW\mathbf{x} + \mathbf{n}, \tag{4.29}$$

where all vectors in Eq. (4.29) have zero mean. We will use as an estimator a linear transformation of the observation vector:

$$\widehat{\mathbf{x}} = H\mathbf{y}. \tag{4.30}$$

The error vector is

$$\begin{aligned} \mathbf{e} &= \mathbf{x} - \widehat{\mathbf{x}} \\ &= \mathbf{x} - H\mathbf{y}. \end{aligned} \tag{4.31}$$

We wish to choose H so as to minimize the variance of the error. Since $\mathbf{e}$ is zero-mean, this is equivalent to minimizing the mean-square-error, given by

$$\begin{aligned} \xi &= \langle \mathbf{e}^T\mathbf{e} \rangle \\ &= \mathrm{Tr}\langle \mathbf{e}\mathbf{e}^T \rangle. \end{aligned} \tag{4.32}$$

Let e_n and x_n be the n^{th} components of the vectors $\mathbf{e}$ and $\mathbf{x}$, and let $\mathbf{h}_n$ be the n^{th} row of H. Then we can write:

$$e_n = x_n - \mathbf{y}^T\mathbf{h}_n. \tag{4.33}$$

We can see that choosing $\mathbf{h}_n$ to minimize $\langle e_n^2 \rangle$, for every n, minimizes ξ. To minimize $\langle e_n^2 \rangle$, we invoke the orthogonality principle (also known as the projection theorem). We choose $\mathbf{h}_n$ so that each component of the error is orthogonal to all components of the observation:

$$\langle e_n \mathbf{y} \rangle = 0. \tag{4.34}$$

Using Eq. (4.33) in Eq. (4.34) yields the equation

$$\langle x_n \mathbf{y} \rangle = \langle \mathbf{y}\mathbf{y}^T \rangle \mathbf{h}_n. \tag{4.35}$$

Solving for $\mathbf{h}_n$ we get

$$\mathbf{h}_n = \langle \mathbf{y}\mathbf{y}^T \rangle^{-1} \langle x_n \mathbf{y} \rangle. \tag{4.36}$$

Thus, the optimum H matrix, H_o, has a value given by the expression

$$H_o = \langle \mathbf{x}\mathbf{y}^T \rangle \langle \mathbf{y}\mathbf{y}^T \rangle^{-1}. \tag{4.37}$$

The covariance matrices in Eq. (4.37) can be computed using Eq. (4.29), yielding

$$\langle \mathbf{x}\mathbf{y}^T \rangle = \langle \mathbf{x}\mathbf{x}^T \rangle (BW)^T, \tag{4.38}$$

$$\langle \mathbf{y}\mathbf{y}^T \rangle = (BW)\langle \mathbf{x}\mathbf{x}^T \rangle (BW)^T + \langle \mathbf{n}\mathbf{n}^T \rangle. \tag{4.39}$$

In computing Eq.'s (4.38) and (4.39) , it was assumed that the object and noise vectors are uncorrelated. Using Eq.'s (4.38) and (4.39) in Eq. (4.37) yields

$$\begin{aligned} H_o &= \langle \mathbf{x}\mathbf{x}^T \rangle (BW)^T \left[(BW)\langle \mathbf{x}\mathbf{x}^T \rangle (BW)^T + \langle \mathbf{n}\mathbf{n}^T \rangle \right]^{-1} \\ &= R_{xx} G^T \left[G\, R_{xx} G^T + R_{nn} \right]^{-1}, \end{aligned} \tag{4.40}$$

where

$$R_{xx} = \langle \mathbf{x}\mathbf{x}^T \rangle, \tag{4.41}$$

$$R_{nn} = \langle \mathbf{n}\mathbf{n}^T \rangle, \tag{4.42}$$

$$G = BW. \tag{4.43}$$

One can apply a matrix inversion lemma to Eq. (4.40) to show that

$$H_o = (G^T R_{nn}^{-1} G + R_{xx}^{-1})^{-1} G^T R_{nn}^{-1}. \tag{4.44}$$

In order to compute the energy spectrum of the error, we need the error covariance matrix. Using Eq. (4.31) we can write

$$\langle \mathbf{e}\mathbf{e}^T \rangle = \langle \mathbf{e}\mathbf{x}^T \rangle - \langle \mathbf{e}\mathbf{y}^T \rangle H_o^T. \tag{4.45}$$

Since we have chosen H_o so that each component of the error vector is orthogonal to the observation vector $\mathbf{y}$, the second term in Eq. (4.45) must be zero. Therefore, we have

$$\begin{aligned} \langle \mathbf{e}\mathbf{e}^T \rangle &= \langle \mathbf{e}\mathbf{x}^T \rangle \\ &= \langle \mathbf{x}\mathbf{x}^T \rangle - H_o \langle \mathbf{y}\mathbf{x}^T \rangle. \end{aligned} \tag{4.46}$$

Using Eq.'s (4.38) and (4.41) - (4.44) in Eq. (4.46) yields

$$\begin{aligned} \langle \mathbf{e}\mathbf{e}^T \rangle &= R_{xx} - \left(G^T R_{nn}^{-1} G + R_{xx}^{-1} \right)^{-1} G^T R_{nn}^{-1} G R_{xx} \\ &= \left(G^T R_{nn}^{-1} G + R_{xx}^{-1} \right)^{-1} \left[\left(G^T R_{nn}^{-1} G + R_{xx}^{-1} \right) R_{xx} - G^T R_{nn}^{-1} G R_{xx} \right] \\ &= \left(G^T R_{nn}^{-1} G + R_{xx}^{-1} \right)^{-1}. \end{aligned} \tag{4.47}$$

To keep things simple we will assume that both the object and noise vectors are white, i.e.,

$$R_{xx} = \sigma_x^2 I, \tag{4.48}$$

$$R_{nn} = \sigma_n^2 I, \tag{4.49}$$

where σ_x^2 and σ_n^2 are the variances of object and noise pixels, respectively. If we define a signal-to-noise ratio as

$$\mathrm{SNR_{REF}} = \sigma_x^2 / \sigma_n^2, \tag{4.50}$$

then Eq. (4.47) can be written

$$\langle \mathbf{e}\mathbf{e}^T \rangle = (G^T G (\mathrm{SNR_{REF}}) + I)^{-1} \sigma_x^2. \tag{4.51}$$

The numerator of the $\mathrm{SNR_{REF}}$ defined by Eq. (4.50) is referenced to the object plane and the denominator is referenced to the image plane. Can we relate this to a similarly-defined SNR totally referenced to the image plane? In general, the answer is no. Such an SNR would vary from pixel to pixel. Rather than attempting to develop a suitable definition for SNR in the image plane, we will instead simply make the following observation. Given a very large object of uniform intensity, the intensity of the image is also uniform and is equal to the intensity of the object scaled by the

MTF evaluated at zero spatial frequency, which is unity. Thus, in the case of a very large object of uniform intensity, the SNR of an image pixel is $\mathrm{SNR_{REF}}$.

We are now in a position to make some computations using the minimum-variance processor and point-spread functions corresponding to the four-subaperture, five-subaperture, and six-subaperture arrays shown in Fig.'s 4.2 – 4.4, respectively. In all cases we will use a pixel spacing δ of

$$\delta = \frac{1}{4}\frac{\lambda}{D}. \tag{4.52}$$

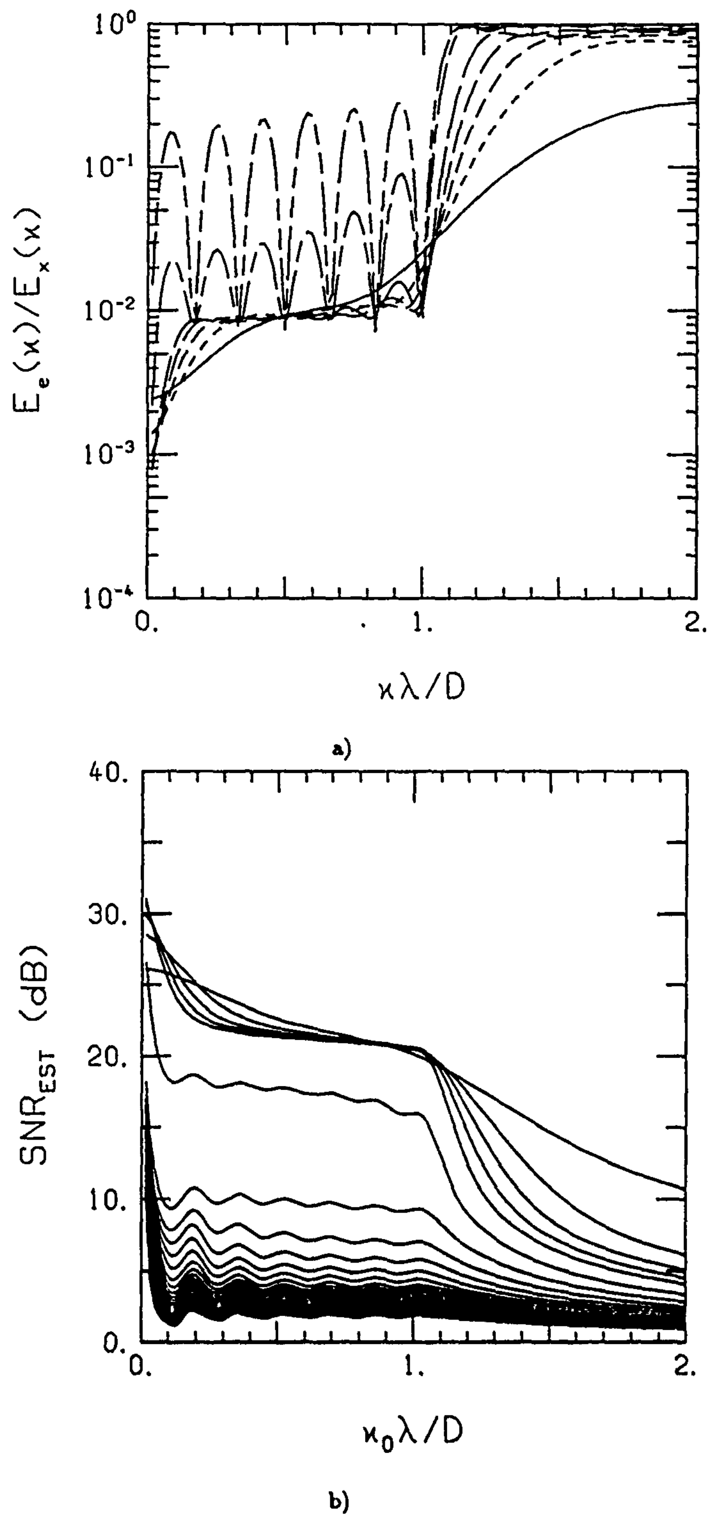
This pixel spacing corresponds to a sampling frequency of $4D/\lambda$ or a Nyquist frequency of $2D/\lambda$. This choice will allow the study of extrapolation superresolution out to a spatial frequency of $2D/\lambda$, twice the diffraction limit of an aperture of length D. The energy spectrum of the object is computed using Eq. (4.48) in Eq. (4.28).

$$E_x(\kappa) = M\delta\sigma_x^2, \tag{4.53}$$

where M is the number of pixels in the object (length of $\mathbf{x}$), δ is the pixels spacing, and σ_x^2 is the variance of an individual object pixel. Note that $M\delta$ is the support of the object. We compute the energy spectrum of the error vector using Eq. (4.51) in Eq. (4.28):

$$E_e(\kappa) = \delta\sigma_x^2 \sum_m \sum_n \left[\left(G^T G \, (\mathrm{SNR_{REF}}) + I \right)^{-1} \right]_{mn}. \tag{4.54}$$

Fig. 4.5a is a plot of the ratio $E_e(\kappa)/E_x(\kappa)$ versus spatial frequency and Fig. 4.5b is a plot of $\mathrm{SNR_{EST}}$ as a function of filter cutoff frequency κ_o using the four-subaperture array of Fig. 4.2 with $d/D = 0.005$, and $\mathrm{SNR_{REF}} = 50dB$ $(10\log_{10}(\sigma_x^2/\sigma_n^2))$. There are seven curves in Fig. 4.5a corresponding to object supports of $1\lambda/D$, $2\lambda/D$, $\cdots$, $7\lambda/D$, with larger support corresponding to larger values of the ratio $E_e(\kappa)/E_x(\kappa)$. Fig. 4.5b contains 32 curves for object supports of $1\lambda/D$, $2\lambda/D$, $\cdots$, $32\lambda/D$. There are several interesting observations that can be made concerning these figures. First, $E_e(\kappa) \leq E_x(\kappa)$ for all spatial frequencies κ. The reason for this is that when the signal at a particular spatial frequency is very small compared to the noise, the minimum-variance estimator "turns off", i.e., $\hat{\mathbf{x}}$ goes to zero, and the error $\mathbf{e}$ converges to the object $\mathbf{x}$. Thus, a value of the ratio close to unity is "bad", and a value very much less than unity is "good". The second point to note is the rapid increase in $E_e(\kappa)/E_x(\kappa)$ and rapid fall-off of $\mathrm{SNR_{EST}}$ for $\kappa > D/\lambda$, i.e., extrapolative super-resolution doesn't work (a not unexpected result). The third point to note, and the most important, is the near uniformity of interpolative super-resolution for object support $\leq 5\lambda/D$. Referring to Fig. 4.2, we see that the minimum-variance algorithm is receiving direct measurements of the object spectrum at only very narrow regions centered about 0, $\pm 1/6D/\lambda$, $\pm 2/6D/\lambda$, $\cdots$, D/λ, but Fig. 4.5a tells us that the object estimates for support $\leq 5\lambda/D$ contain approximately the same quality of spectral information throughout the band below D/λ as the direct measurements. Thus, the processor is "filling in the gaps" with apparent ease. This behavior manifests itself in Fig. 4.5b also, where we see that the first five curves group together at a near uniform $\mathrm{SNR_{EST}}$ of about $22dB$ out to $\kappa_o = D/\lambda$. When object support exceeds $5\lambda/D$, we note from Fig. 4.5a the processors inability to "fill in the gaps" between measurements and, from Fig. 4.5b, the rapid drop in $\mathrm{SNR_{EST}}$ over the range $\kappa_o \leq D/\lambda$ with increasing object support. We can illustrate this behavior rather dramatically by taking a vertical slice through Fig. 4.5b at $\kappa_o = D/\lambda$, and plotting $\mathrm{SNR_{EST}}$ versus object support. This result is shown in Fig. 4.6. The curve speaks for itself. We note only that the "cliff" of Fig. 4.6 is located at a support level approximately equal to the reciprocal of the spacing between "islands" in the MTF of Fig. 4.2. The question arises: have we established a rule? That is, interpolative super-resolution "works" as long as object support is less than the reciprocal of the spacing between nonzero "islands" in the array MTF. By examining other cases, we will see that this is indeed the rule. Furthermore, we will see that even if the rule is violated by only a small amount, severe performance penalties are the result.



b)

Figure 4.5.

Ratio of the energy spectrum of the error to the energy spectrum of the object versus spatial frequency κ (a) and signal-to-noise ratio of the estimate versus low-pass filter cutoff frequency κ_0 (b) using the minimum-variance processor and the four-subaperture array with $SNR_{REF} = 50dB$ and $d/D = 0.005$. The seven curves of (a) correspond to object supports of $1\lambda/D, 2\lambda/D, \ldots, 7\lambda/D$, with $E_e(\kappa)/E_x(\kappa)$ larger for larger support values. There are 32 curves in (b) corresponding to object supports of $1\lambda/D, 2\lambda/D, \ldots, 32\lambda/D$, with larger SNR_{EST} for smaller support values.

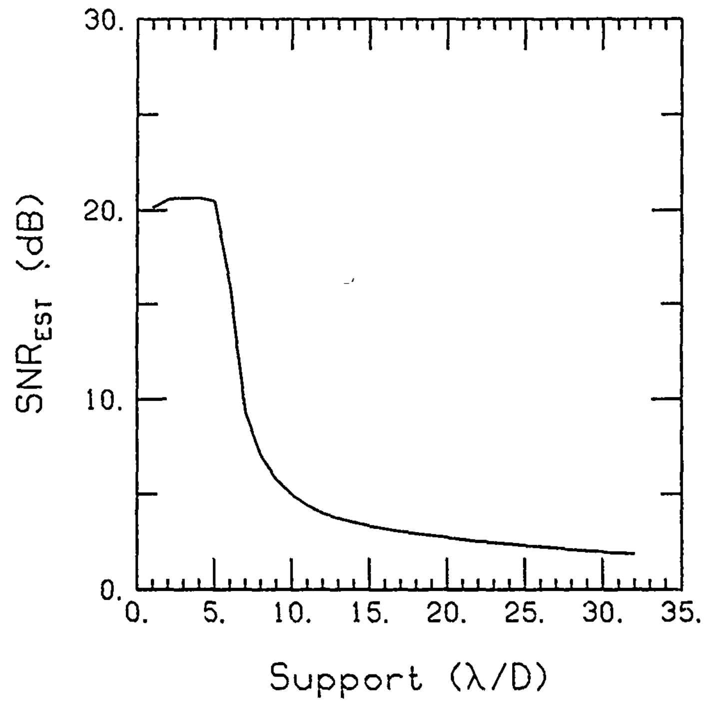


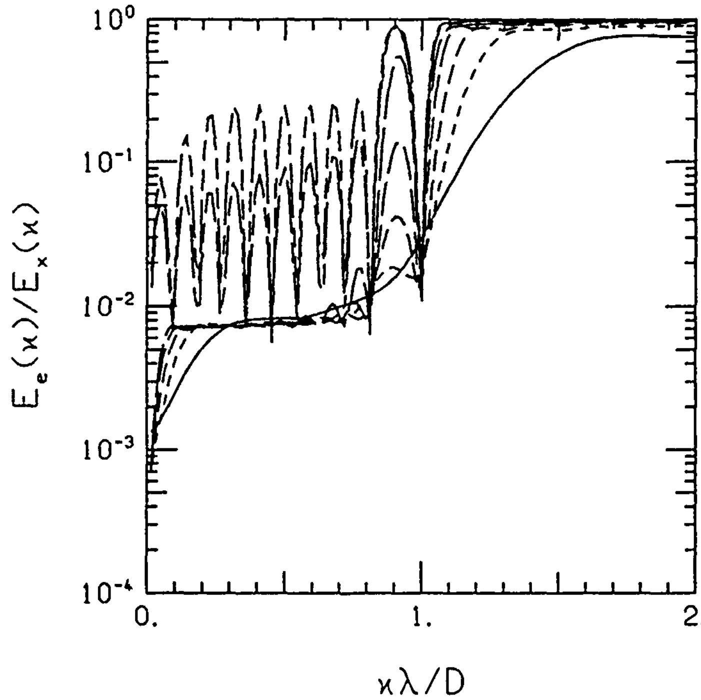
Figure 4.6.

Signal-to-noise ratio of the estimate versus object support using the minimum-variance processor and the four-subaperture array, with $SNR_{REF} = 50dB$, $d/D = 0.005$, and $\kappa_o = 1D/\lambda$.

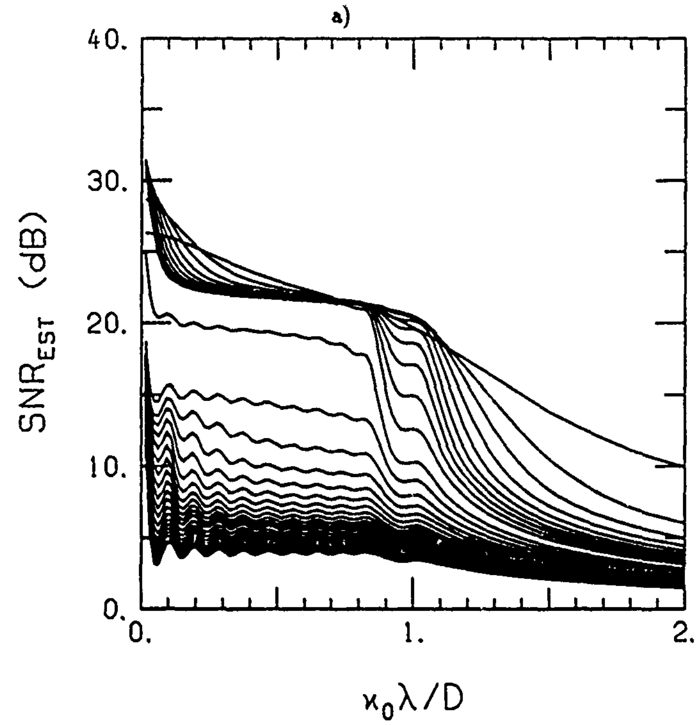
Performance plots for the minimum-variance processor and the five-subaperture array of Fig. 4.3 are shown in Fig. 4.7. Again $d/D = 0.005$ and $SNR_{REF} = 50dB$. There are again seven curves in Fig. 4.7(a), this time corresponding to object supports of $2\lambda/D$, $4\lambda/D$, $\cdots$, $14\lambda/D$, with larger support corresponding to larger values of $E_e(\kappa)/E_x(\kappa)$. Fig. 4.7(b) again contains 32 curves corresponding to object supports of $1\lambda/D$, $2\lambda/D$, $\cdots$, $32\lambda/D$. Again, note the near uniformity of interpolative super-resolution for the first five curves of Fig. 4.7(a), this time over a spatial frequency range $|\kappa| \leq 0.81D/\lambda$, the "useful" range of the five-subaperture array (see Fig. 4.3(b)). In this case, the first five curves correspond to object support $\leq 10\lambda/D$.

Performance plots for the six-subaperture array of Fig. 4.4 are shown in Fig. 4.8. The results are so similar to those of the four and five subaperture cases that we will not further elaborate, other than to point out that the "useful" range of spatial frequencies for good interpolative super-resolution performance for objects of support $\leq 15\lambda/D$ is $|\kappa| \leq 0.76D/\lambda$, the same as the "useful" range for the six-subaperture array.

If we take vertical slices through the plots of Fig.'s 4.7(b) and 4.8(b) at the upper limit of the "useful" frequency range of their corresponding arrays, $\kappa_o = 0.81D/\lambda$ and $\kappa_o = 0.76D/\lambda$, respectively, we have the curves of Fig.'s 4.9(b) and 4.9(c). For convenience, Fig. 4.6, the corresponding curve for the four-subaperture case, is reproduced in Fig. 4.9(a). In each case, the precipitous drop in performance occurs at the point where object support exceeds the reciprocal of the spacing between "islands" in the MTF for the corresponding array. One might argue that this result is "obvious" from the sampling theorem. That is, the sparse array gives us uniformly spaced samples of the Fourier transform of the object intensity distribution throughout the "useful" range of the array. Thus, it should be possible to reconstruct the object intensity distribution from these samples out to a resolution limit equal to the highest spatial frequency of the "useful" range, so long as the support of the object is less than the reciprocal of the sample spacing. However, this argument, although a useful viewpoint in developing an intuitive understanding of the results, gives no indication of how



a)



b)

Figure 4.7.

Ratio of the energy spectrum of the error to the energy spectrum of the object versus spatial frequency κ (a) and signal-to-noise ratio of the estimate versus low-pass filter cutoff frequency κ_o (b) using the minimum-variance processor and the five-subaperture array with $SNR_{REF} = 50dB$ and $d/D = 0.005$. The seven curves of (a) correspond to object supports of $2\lambda/D$, $4\lambda/D$, ..., $14\lambda/D$, with $E_e(\kappa)/E_x(\kappa)$ larger for larger support values. There are 32 curves in (b) corresponding to object supports of $1\lambda/D$, $2\lambda/D$, ..., $32\lambda/D$, with larger SNR_{EST} for smaller support values.

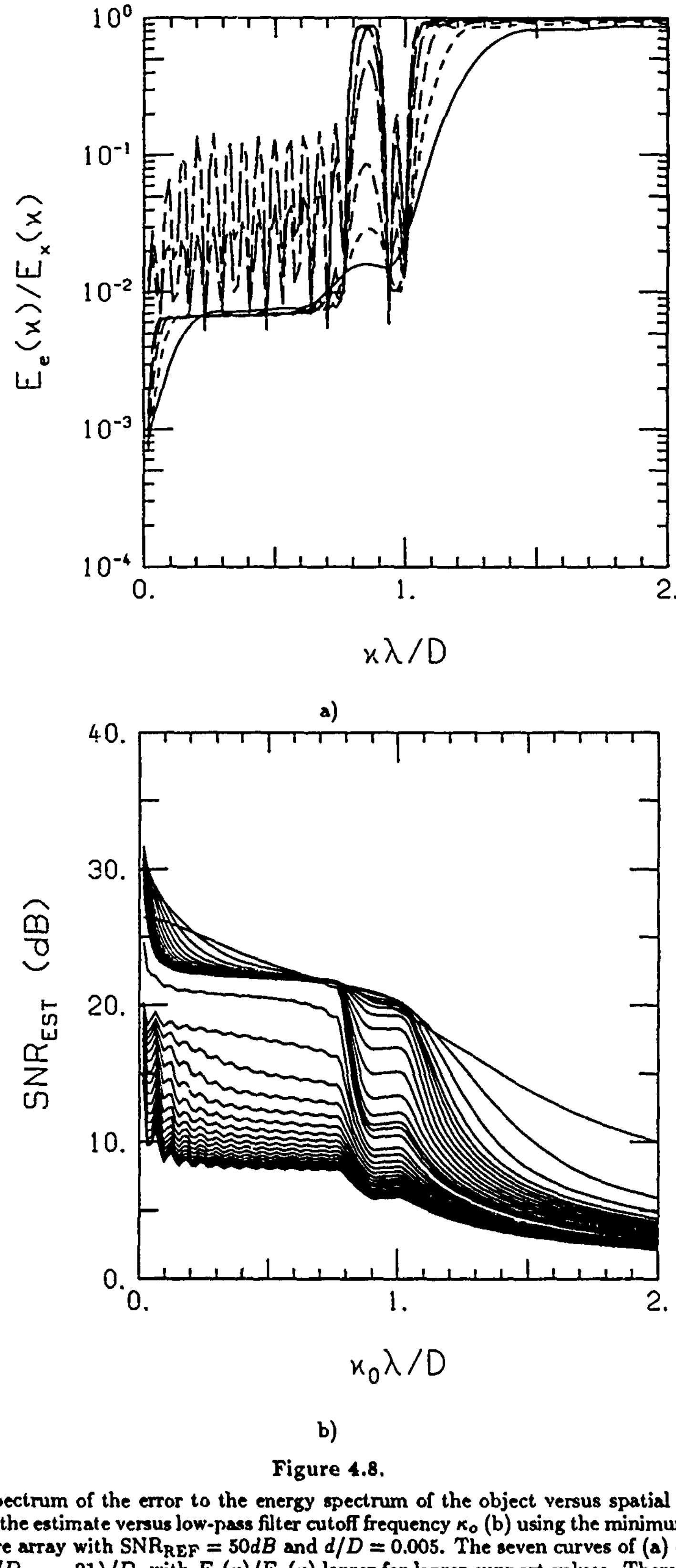
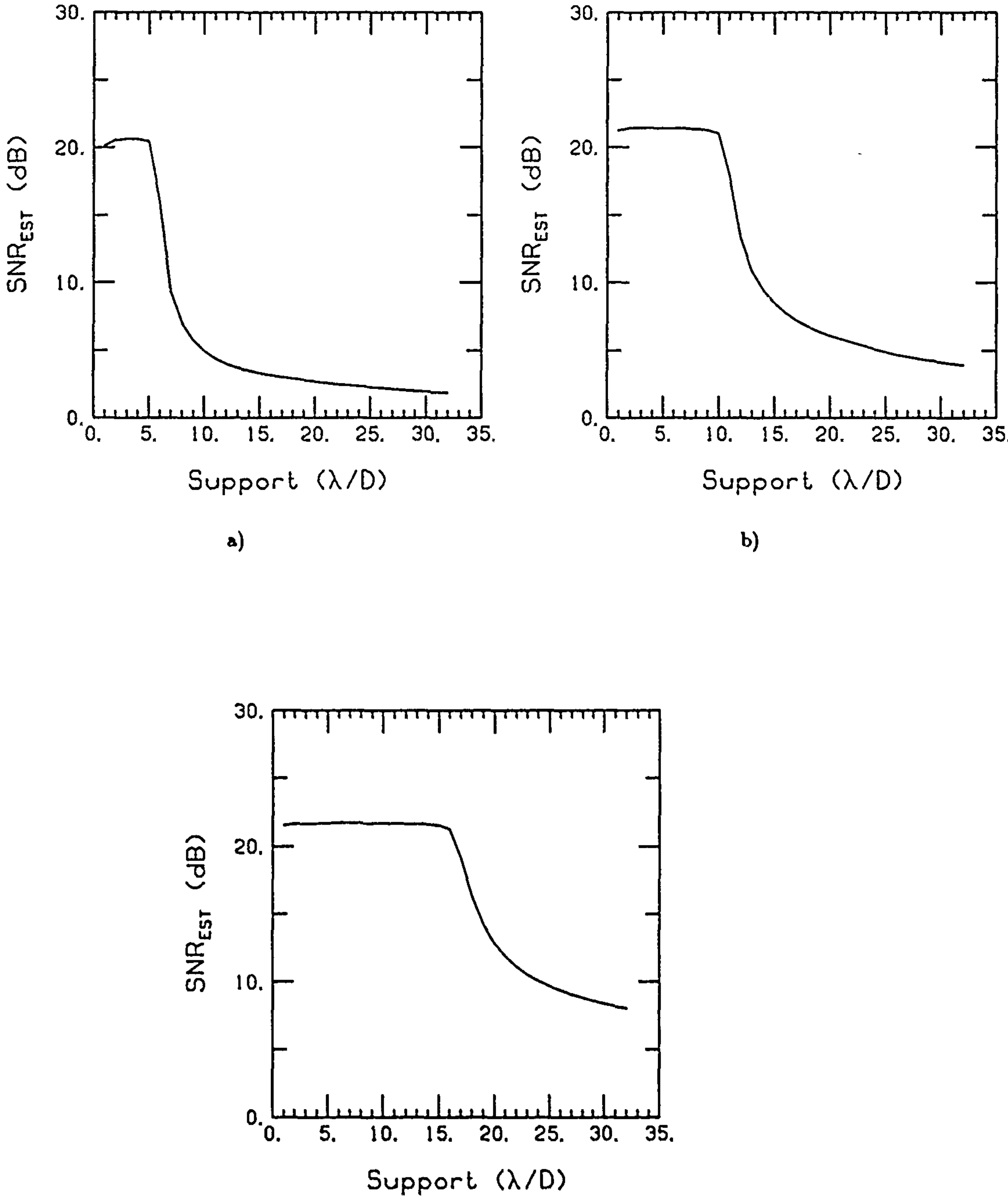


Figure 4.8.

Ratio of the energy spectrum of the error to the energy spectrum of the object versus spatial frequency κ (a) and signal-to-noise ratio of the estimate versus low-pass filter cutoff frequency κ_o (b) using the minimum-variance processor and the six-subaperture array with $\mathrm{SNR_{REF}} = 50dB$ and $d/D = 0.005$. The seven curves of (a) correspond to object supports of $3\lambda/D$, $6\lambda/D, \ldots, 21\lambda/D$, with $E_e(\kappa)/E_x(\kappa)$ larger for larger support values. There are 32 curves in (b) corresponding to object supports of $1\lambda/D$, $2\lambda/D, \ldots, 32\lambda/D$, with larger $\mathrm{SNR_{EST}}$ for smaller support values.



a) b) c)

Figure 4.9.

Signal-to-noise ratio of the estimate versus object support using the minimum-variance processor, $\text{SNR}_{\text{REF}} = 50dB$, and $d/D = 0.005$. Fig.'s (a), (b), and (c) correspond respectively, to the four-subaperture array with $\kappa_o = 1D/\lambda$, the five-subaperture array with $\kappa_o = 0.81D/\lambda$, and the six-subaperture array with $\kappa_o = 0.76D/\lambda$.

severe the penalty is for violating the support constraint nor does it give any indication of how much SNR loss will occur within the "useful" spatial frequency range of the array.

4.5. Unweighted Least-Squares Processor

In computing the results of Section 4.4, we assumed knowledge of the mean and covariance of the object and noise vectors and a finite-support constraint on the object. A probably more realistic situation is one where we are presented with an image and given the MTF of the optical system, and we are required to find the object which best explains the image in some sense, given no statistical knowledge. In this section we will use a least-squares criterion for deciding which object "best" explains the image, i.e., we pick the object vector estimate $\widehat{\mathbf{x}}$ which minimizes

$$\epsilon = ||\mathbf{y} - G\widehat{\mathbf{x}}||^2. \tag{4.55}$$

Here we use the same one-dimensional optical model as that described in Section 4.2, with $\mathbf{y}$ the vector of image pixels, $\mathbf{x}$ the vector of object pixels, and $G = BW$ a matrix containing the system point-spread function and the object support information. Carrying out the operation indicated in Eq. (4.55) yields

$$\begin{aligned} \epsilon &= (\mathbf{y} - G\widehat{\mathbf{x}})^T(\mathbf{y} - G\widehat{\mathbf{x}}) \\ &= \mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T G\widehat{\mathbf{x}} + \widehat{\mathbf{x}}^T G^T G\widehat{\mathbf{x}}. \end{aligned} \tag{4.56}$$

The gradient of ϵ with respect to $\widehat{\mathbf{x}}$ is

$$\nabla\epsilon = -2G^T\mathbf{y} + 2G^T G\widehat{\mathbf{x}}, \tag{4.57}$$

where here the gradient is taken to be a column vector. Setting $\Delta\epsilon$ equal to zero and solving for $\widehat{\mathbf{x}}$ yields

$$\widehat{\mathbf{x}} = (G^T G)^{-1}G^T\mathbf{y}. \tag{4.58}$$

Since G has full column rank, $G^T G$ is nonsingular and its inverse is well defined. The error vector with this estimate is

$$\begin{aligned} \mathbf{e} &= \mathbf{x} - \widehat{\mathbf{x}} \\ &= \mathbf{x} - (G^T G)^{-1}G^T\mathbf{y}. \end{aligned} \tag{4.59}$$

Using $\mathbf{y}$ from Eq. (4.5) in Eq. (4.59), with $G = BW$, yields

$$\begin{aligned} \mathbf{e} &= \mathbf{x} - (G^T G)^{-1}G^T(G\mathbf{x} + \mathbf{n}) \\ &= (G^T G)^{-1}G^T\mathbf{n}. \end{aligned} \tag{4.60}$$

The covariance matrix of the error vector is

$$\begin{aligned} \langle\mathbf{e}\mathbf{e}^T\rangle &= (G^T G)^{-1}G^T\langle\mathbf{n}\mathbf{n}^T\rangle G(G^T G)^{-1} \\ &= (G^T G)^{-1}G^T R_{nn} G(G^T G)^{-1}. \end{aligned} \tag{4.61}$$

As in the minimum-variance case, we will assume that the object and noise vectors are white, i.e.,

$$R_{xx} = \sigma_x^2 I, \tag{4.62}$$

$$R_{nn} = \sigma_n^2 I, \tag{4.63}$$

and

$$\mathrm{SNR}_{\mathrm{REF}} = \sigma_x^2/\sigma_n^2. \tag{4.64}$$

Then

$$\begin{aligned}\langle \mathbf{e}\mathbf{e}^T \rangle &= (G^T G)^{-1}\sigma_n^2 \\ &= \left(G^T G(\mathrm{SNR}_{\mathrm{REF}})\right)^{-1}\sigma_x^2\end{aligned} \tag{4.65}$$

We write the error covariance matrix in this way to emphasis its similarity to the error covariance matrix for the minimum-variance processor (see Eq. (4.51)). In fact, we see that as $\mathrm{SNR}_{\mathrm{REF}}$ increases the two matrices become one. In other words, for large enough $\mathrm{SNR}_{\mathrm{REF}}$, the performance of unweighted least-squares approaches that of minimum-variance.

We compute the energy spectrum of the error vector using Eq. (4.65) in Eq. (4.28):

$$E_e(\kappa) = \delta\sigma_x^2 \sum_m \sum_n \left[(G^T G(\mathrm{SNR}_{\mathrm{REF}}))^{-1}\right]_{mn}. \tag{4.66}$$

The energy spectrum of the object is given by Eq. (4.53). The pixel spacing we will use is

$$\delta = \tfrac{1}{2}\lambda/D, \tag{4.67}$$

yielding a Nyquist rate of D/λ. There are two reasons for changing the pixel spacing from $\frac{1}{4}\lambda/D$ used in the minimum-variance processor. First, we are no longer interested in studying extrapolative super-resolution performance and thus a Nyquist rate of D/λ is adequate. Second, the inclusion of an extrapolative super-resolution region creates a very large eigenvalue spread in $G^T G$, making it very difficult to numerically evaluate $E_e(\kappa)$ with the numerical precision at our disposal. This was not a problem with the minimum-variance processor because of the stabilizing influence of the identity matrix in Eq. (4.54).

Except for pixel spacing, all of the results to follow were computed using the same parameters as with the minimum-variance processor, i.e., $d/D = 0.005$ and $\mathrm{SNR}_{\mathrm{REF}} = 50dB$. $E_e(x)/E_x(\kappa)$ and $\mathrm{SNR}_{\mathrm{EST}}(\kappa_o)$ are shown in Fig.'s 4.10, 4.11, and 4.12 using the four, five, and six-subaperture array, respectively. Corresponding plots of $\mathrm{SNR}_{\mathrm{EST}}$ versus objec. support at the appropriate value of κ_o are shown in Fig. 4.13. Here we see similar features to that of the minimum-variance processor: near uniform interpolative super-resolution with object support less than the reciprocal of the "island" spacing in the MTF and a very rapid drop in performance when object support exceeds this value. Note that the least-squares processor is not "smart" enough to "turn-off" when the object estimate gets to noisy, thus allowing $\mathrm{SNR}_{\mathrm{EST}}$ to drop below zero dB. For purposes of comparison, Fig. 4.14 shows the $\mathrm{SNR}_{\mathrm{EST}}$ versus support curves for minimum-variance and least-squares superimposed.

4.6. Discussion

In presenting the minimum-variance and least-squares results, we have ignored certain subtleties for the sake of clarity. For example, the number of columns in the $G = BW$ matrix is determined by the support of the object. How was the number of rows determined? Since the MTF in our optical model is of finite extent, the point-spread function must be infinite, and therefore so is the image of the object. In any practical imaging system, one obviously must truncate the image. In our model, the number of pixels in the truncated image is determined by the number of rows of G. In general, the larger the number of rows of G, the better is the object reconstruction. However, a point of diminishing returns is reached whereby increasing the number of rows of G does not significantly increase performance. At present, we have no theory to predict when this point is reached. However, we determined imperically that an image line of length 256 λ/D was essentially equivalent to an infinite image line for all the cases we have presented in this report, and that is the length that was used for all cases.

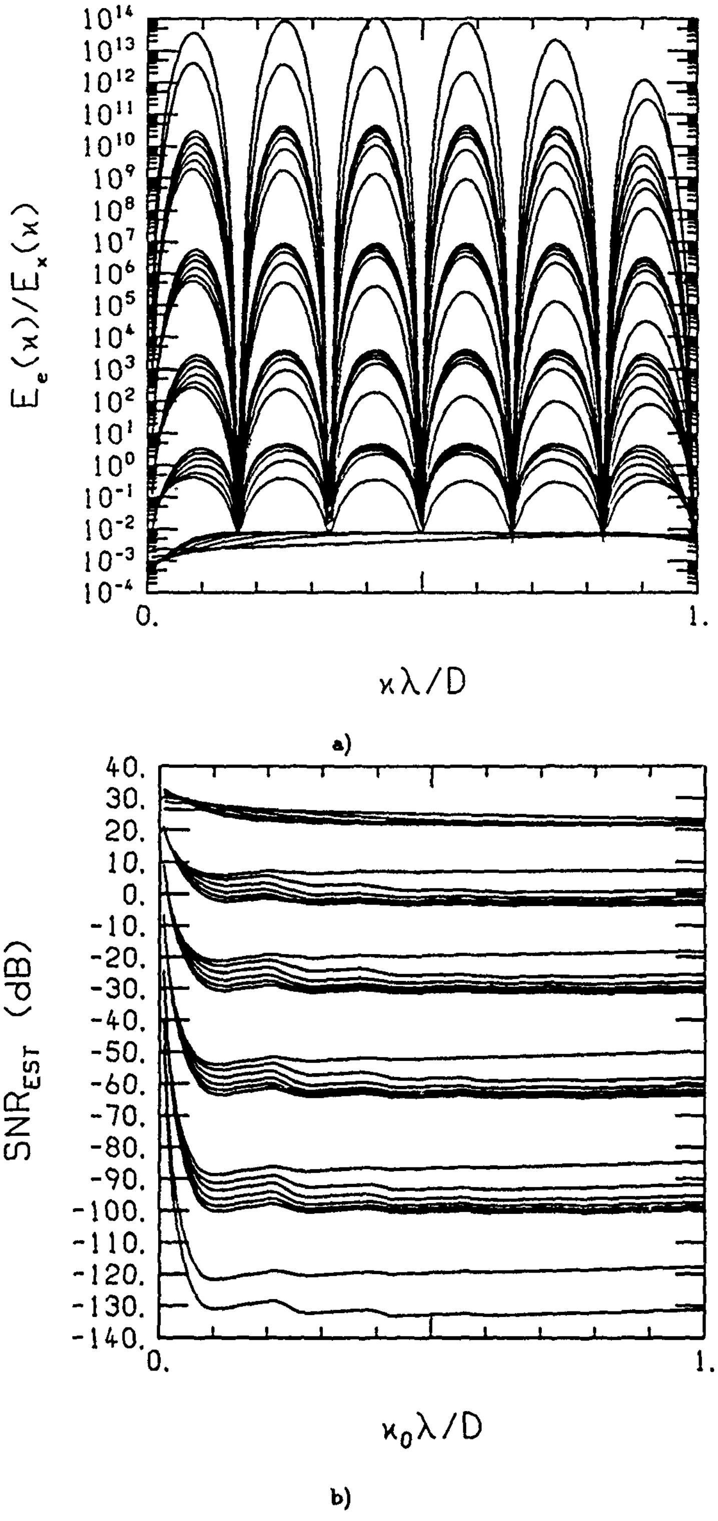
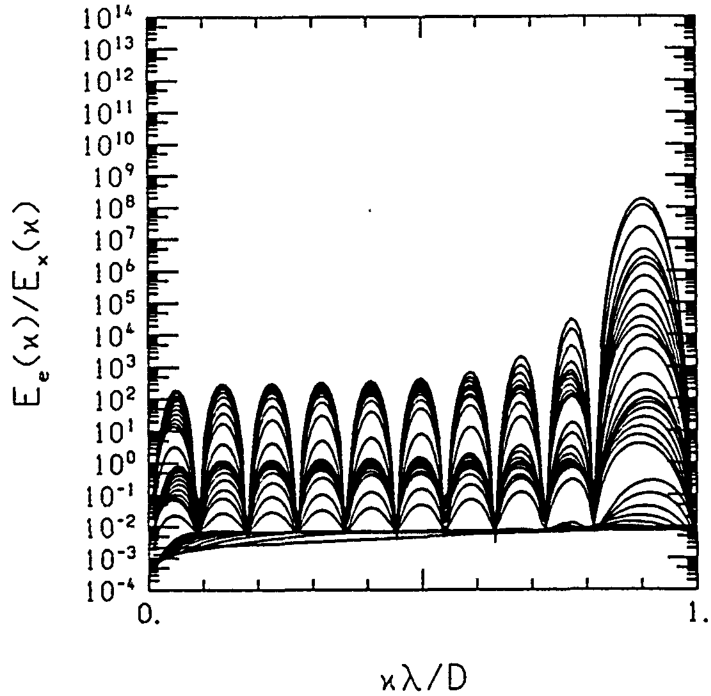
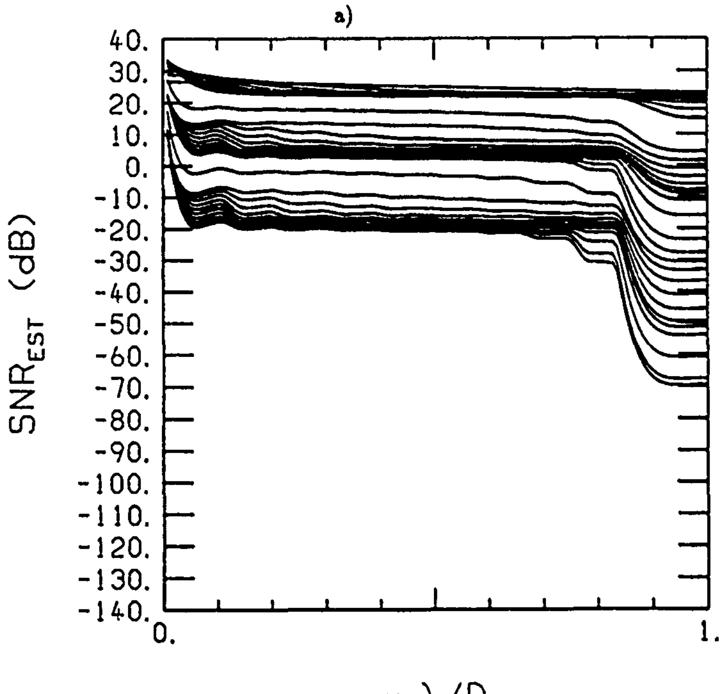


Figure 4.10.

Ratio of the energy spectrum of the error to the energy spectrum of the object versus spatial frequency κ (a) and signal-to-noise ratio of the estimate versus low-pass filter cutoff frequency κ_o (b) using the unweighted least-squares processor and the four-subaperture array with $SNR_{REF} = 50dB$ and $d/D = 0.005$. The 32 curves of (a) correspond to object supports of $1\lambda/D$, $2\lambda/D, \ldots, 32\lambda/D$, with $E_e(\kappa)/E_x(\kappa)$ larger for larger support values. There are 32 curves in (b) corresponding to object supports of $1\lambda/D$, $2\lambda/D, \ldots, 32\lambda/D$, with larger SNR_{EST} for smaller support values.



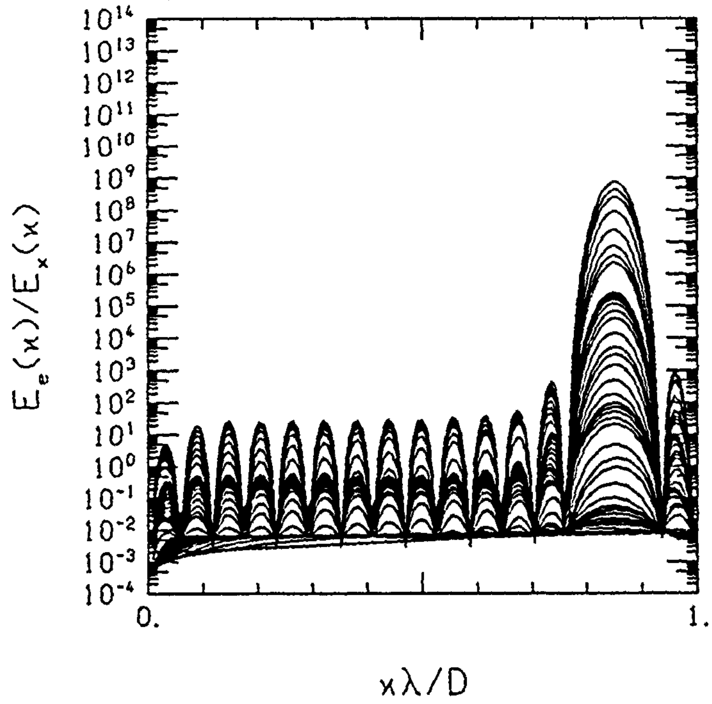
a)



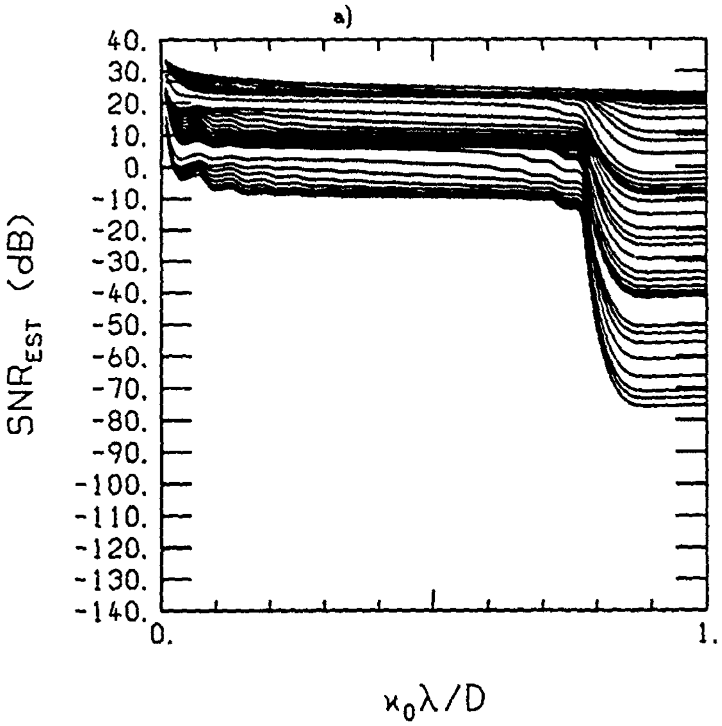
b)

Figure 4.11.

Ratio of the energy spectrum of the error to the energy spectrum of the object versus spatial frequency κ (a) and signal-to-noise ratio of the estimate versus low-pass filter cutoff frequency κ_o (b) using the unweighted least-squares processor and the five-subaperture array with $\mathrm{SNR_{REF}} = 50dB$ and $d/D = 0.005$. The 32 curves of (a) correspond to object supports of $1\lambda/D$, $2\lambda/D, \ldots, 32\lambda/D$, with $E_e(\kappa)/E_x(\kappa)$ larger for larger support values. There are 32 curves in (b) corresponding to object supports of $1\lambda/D$, $2\lambda/D, \ldots, 32\lambda/D$, with larger $\mathrm{SNR_{EST}}$ for smaller support values.



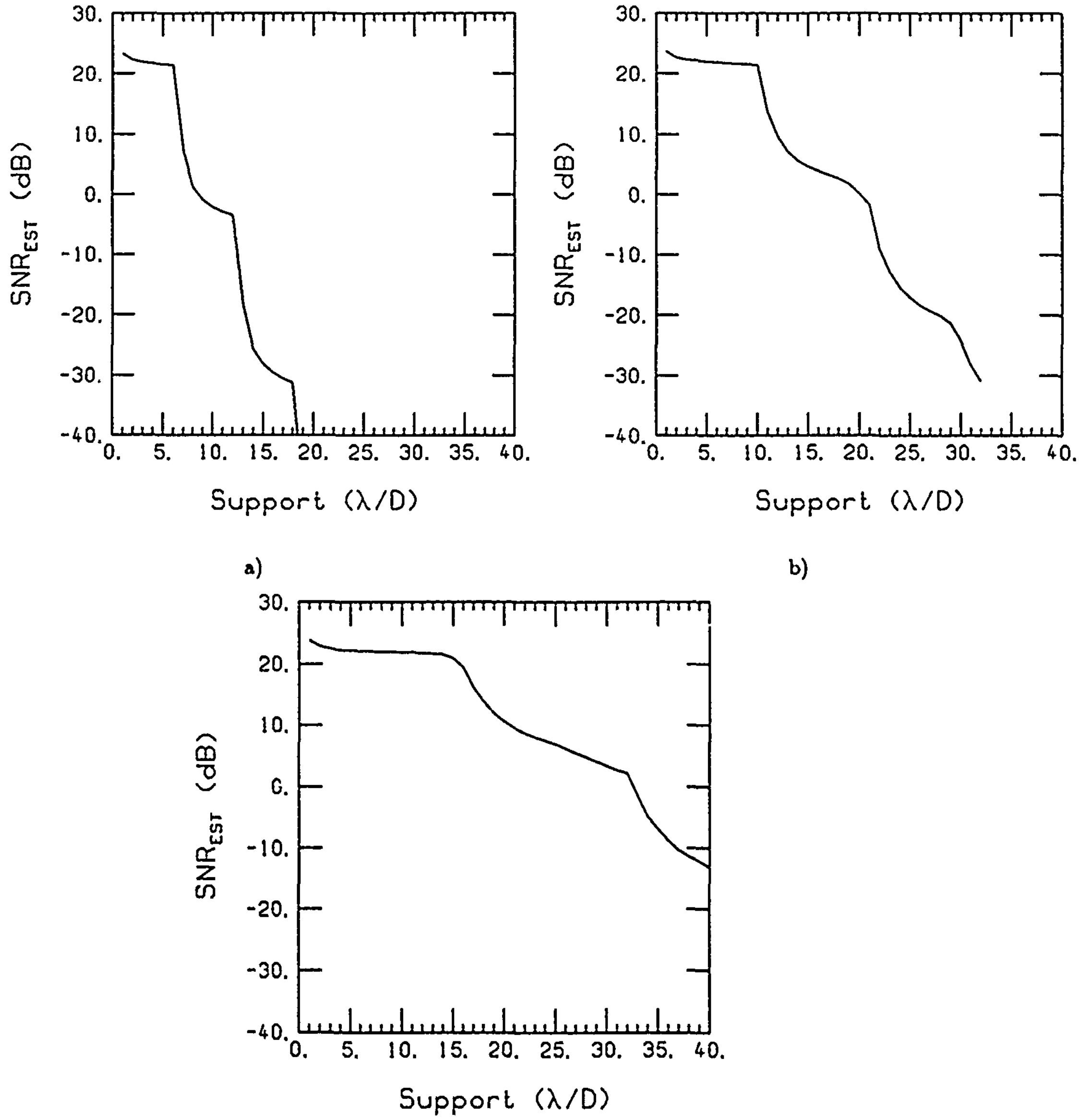
a)



b)

Figure 4.12.

Ratio of the energy spectrum of the error to the energy spectrum of the object versus spatial frequency κ (a) and signal-to-noise ratio of the estimate versus low-pass filter cutoff frequency κ_0 (b) using the unweighted least-squares processor and the six-subaperture array with $SNR_{REF} = 50dB$ and $d/D = 0.005$. The 32 curves of (a) correspond to object supports of $1\lambda/D$, $2\lambda/D$, ..., $32\lambda/D$, with $E_e(\kappa)/E_x(\kappa)$ larger for larger support values. There are 32 curves in (b) corresponding to object supports of $1\lambda/D$, $2\lambda/D$, ..., $32\lambda/D$, with larger SNR_{EST} for smaller support values.



a) b)

c)

Figure 4.13.

Signal-to-noise ratio of the estimate versus object support using the unweighted least-squares processor, $SNR_{REF} = 50dB$, and $d/D = 0.005$. Fig.'s (a), (b), and (c) correspond respectively, to the four-subaperture array with $\kappa_o = 1D/\lambda$, the five-subaperture array with $\kappa_o = 0.81D/\lambda$, and the six-subaperture array with $\kappa_o = 0.76D/\lambda$.

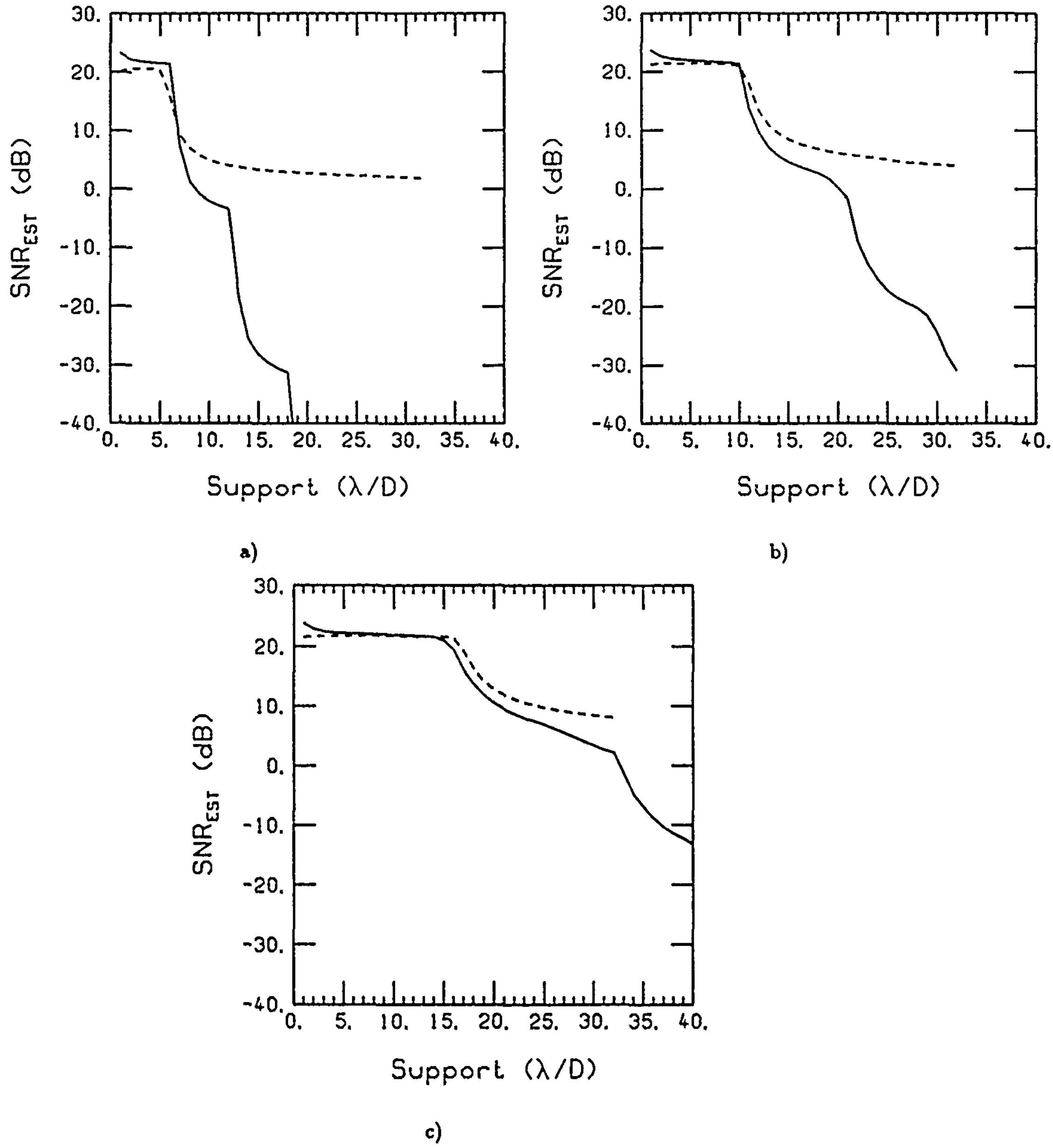


Figure 4.14.

Signal-to-noise ratio of the estimate versus object support using the unweighted least-squares processor (solid line) or the minimum-variance processor (dashed line), $\mathrm{SNR}_{\mathrm{REF}} = 50dB$, and $d/D = 0.005$. Fig.'s (a), (b), and (c) correspond respectively, to the four-subaperture array with $\kappa_o = 1D/\lambda$, the five-subaperture array with $\kappa_o = 0.81D/\lambda$, and the six-subaperture array with $\kappa_o = 0.76D/\lambda$.

Experiments were conducted using values of d/D other than 0.005, which is the value used throughout this report. We found, for small values of d/D, that $\mathrm{SNR_{EST}}$ to the left of the "cliff" was proportional to d/D, as long as the size of the image line was adjusted in each case so that the image line was effectively "infinity" from a performance standpoint.

All the results presented in this report used $\mathrm{SNR_{REF}} = 50\ dB$. However, many experiments were conducted using other values of this parameter. As one would expect, $\mathrm{SNR_{EST}}$ tracked $\mathrm{SNR_{REF}}$. In fact, we can see from Eq. (4.66) that $\mathrm{SNR_{EST}}$ with the least-squares processor is directly proportional to $\mathrm{SNR_{REF}}$. Because of the identity matrix in Eq. (4.54), we can see that this is not the case using the minimum-variance processor. However, at $\mathrm{SNR_{REF}} = 50\ dB$ and above, the effect of the identity matrix in Eq. (4.54) is negligible for object support sizes to the left of the "cliff" edge. We can see this from Fig. 4.14. At $\mathrm{SNR_{REF}}$ below 50 dB, the identity matrix in Eq. (4.54) begins to dominate, and minimum-variance performance is better than that of least-squares. However, at this level of $\mathrm{SNR_{REF}}$, $\mathrm{SNR_{EST}}$ to the left of the "cliff" drops below 20 dB (a voltage ratio less than 10), so that the usefulness of the object reconstruction is open to question. The conclusion is, at least for the cases that we considered, that minimum-variance offers no apparent advantage over least-squares.

Chapter 5

A Random Process with Finite Support: Autocorrelation Function of Its Fourier Transform and Energy and Power Spectral Densities

(originally issued as TR-1042)

5.1. Introduction

It is a well-known property of a wide-sense stationary random process that its Fourier transform is delta-correlated. This is not the case for the Fourier transform of a random processes with finite support. In this report we compute the autocorrelation function of the Fourier transform of a particular random process with finite support. Our particular process is modeled as a windowed wide-sense-stationary random process. We use this autocorrelation function to derive the energy spectral density and power spectral density of the process. We present the results for both continuous (Section 5.2) and discrete (Section 5.3) random processes.

5.2. Continuous Case

5.2.1 Autocorrelation Function of Fourier Transform

Let $z(r)$ be a wide-sense-stationary random process with zero-mean, autocorrelation function $R_z(r)$, and power spectral density $\Phi_z(\kappa)$, defined by the equations

$$R_z(r) = \langle z(r'+r)z(r')\rangle\,, \tag{5.1}$$

$$\Phi_z(\kappa) = \int dr\, \exp(-2\pi i\kappa r)R_z(r). \tag{5.2}$$

We shall let

$$x(r) = w(r)z(r), \tag{5.3}$$

where $w(r)$ is a rectangular window function defining the support of $x(r)$, given by

$$w(r) = \begin{cases} 1, & |r| \le L/2 \\ 0, & \text{else.} \end{cases} \tag{5.4}$$

The Fourier transform of $x(r)$ is

$$\begin{aligned} \tilde{x}(\kappa) &= \int dr\, x(r)\exp[-2\pi i r\kappa] \\ &= \int dr\, w(r)z(r)\exp[-2\pi i r\kappa]. \end{aligned} \tag{5.5}$$

Using Eq. (5.5), we form the correlation between two Fourier coefficients of $x(r)$:

$$\begin{aligned} \langle \tilde{x}(\kappa+\Delta\kappa)\tilde{x}^*(\kappa)\rangle &= \iint dr\, dr'\, w(r)w(r')\langle z(r)z^*(r')\rangle \\ &\qquad \times \exp\left[-2\pi i(r-r')\kappa - 2\pi i r\Delta\kappa\right] \\ &\doteq \iint dr\, dr'\, w(r)w(r')R_z(r-r') \\ &\qquad \times \exp\left[-2\pi i(r-r')\kappa - 2\pi i r\Delta\kappa\right]. \end{aligned} \tag{5.6}$$

Throughout this report, angle brackets denote ensemble average. We now make the following change of variables in Eq. (5.6):

$$u = r - r', \tag{5.7}$$

$$v = (r+r')/2, \tag{5.8}$$

$$r = v + u/2, \tag{5.9}$$

$$r' = v - u/2, \tag{5.10}$$

$$dr\, dr' = du\, dv. \tag{5.11}$$

Then we have

$$\langle \tilde{x}(\kappa+\Delta\kappa)\tilde{x}^*(\kappa)\rangle = \iint du\,dv\,w(v+u/2)\,w(v-u/2)\,R_x(u)$$
$$\times \exp[-2\pi i\kappa u]\exp[-2\pi i\Delta\kappa\,(v+u/2)]$$
$$= \int du\,R_x(u)F(u,\Delta\kappa)\exp[-2\pi iu\,(\kappa+\Delta\kappa/2)]\,, \tag{5.12}$$

where

$$F(u,\Delta\kappa) = \int dv\,w(v+u/2)\,w(v-u/2)\exp[-2\pi i\Delta\kappa v]$$
$$= \begin{cases} (L-|u|)\mathrm{sinc}\,[(L-|u|)\Delta\kappa]\,, & |u|\le L \\ 0, & |u|>L, \end{cases} \tag{5.13}$$

with

$$\mathrm{sinc}(x) = \sin(\pi x)/\pi x. \tag{5.14}$$

Substituting Eq. (5.13) into Eq. (5.12) yields

$$\langle \tilde{x}(\kappa+\Delta\kappa)\tilde{x}^*(\kappa)\rangle = \int_{-L}^{L} du\,R_x(u)(L-|u|)\mathrm{sinc}[(L-|u|)\Delta\kappa]\exp[-2\pi iu\,(\kappa+\Delta\kappa/2)]\,. \tag{5.15}$$

Given any $R_x(u)$, Eq. (5.15) can be used to numerically evaluate the correlation of Fourier coefficients of $x(r)$. However, if we assume that $R_x(u)$ is a "narrow" function compared to the interval $|u|\le L$, i.e.,

$$R_x(u)\simeq 0, \qquad |u|>\epsilon, \qquad \epsilon \ll L, \tag{5.16}$$

then Eq. (5.15) can be approximated by

$$\langle \tilde{x}(\kappa+\Delta\kappa)\tilde{x}^*(\kappa)\rangle \simeq L\,\mathrm{sinc}(L\Delta\kappa)\int du\,R_x(u)\exp[-2\pi iu\,(\kappa+\Delta\kappa/2)]$$
$$\simeq L\,\mathrm{sinc}(L\Delta\kappa)\Phi_x\,(\kappa+\Delta\kappa/2)\,. \tag{5.17}$$

If $x(r)$ is white, i.e.,

$$R_x(r) = \Phi_x\delta(r), \tag{5.18}$$

then we have Eq. (5.17) with equality,

$$\langle \tilde{x}(\kappa+\Delta\kappa)\tilde{x}^*(\kappa)\rangle = L\,\mathrm{sinc}(L\Delta\kappa)\,\Phi_x. \tag{5.19}$$

Note from Eq. (5.17) and (5.19) that the Fourier coefficients of $x(r)$ are uncorrelated at $\Delta\kappa$ equal to an integer multiple of $1/L$, the reciprocal of the support of $x(r)$, if $R_x(r)$ satisfies the assumption given by Eq. (5.16).

5.2.2 Energy Spectral Density and Power Spectral Density of $x(r)$

The two-sided energy spectral density of any random process $x(r)$ of finite support is given by

$$E_x(\kappa) = \langle|\tilde{x}(\kappa)|^2\rangle. \tag{5.20}$$

If $x(r)$ is given by Eq. (5.3), then, from Eq. (5.15), we have

$$E_x(\kappa) = \int_{-L}^{L} du\,R_x(u)(L-|u|)\exp[-2\pi iu\kappa]. \tag{5.21}$$

We recognize Eq. (5.21) as the Fourier transform of the product of $R_z(u)$ with a triangle function. We can therefore write Eq. (5.21) as the convolution of the Fourier transforms of these two functions:

$$E_x(\kappa) = L^2 \int d\kappa' \, \Phi_z(\kappa - \kappa')\mathrm{sinc}^2(L\kappa'). \tag{5.22}$$

The power spectral density $\Phi_x(\kappa)$ of $x(r)$ can be defined as

$$\Phi_x(\kappa) = E_x(\kappa)/L. \tag{5.23}$$

From Eq. (5.21) and (5.22) we then have

$$\Phi_x(\kappa) = \frac{1}{L}\int_{-L}^{L} du \, R_z(u)(L - |u|)\exp[-2\pi i u\kappa], \tag{5.24}$$

or

$$\Phi_x(\kappa) = L \int d\kappa' \, \Phi(\kappa - \kappa')\mathrm{sinc}^2(L\kappa'). \tag{5.25}$$

If $R_z(r)$ satisfies the assumption given by Eq. (5.16), then the power spectral density of $x(r)$ is approximated by

$$\Phi_x(\kappa) \simeq \Phi_z(\kappa). \tag{5.26}$$

We see from Eq. (5.26) that when the assumption given by Eq. (5.16) is satisfied, the power spectral density of $x(r)$ is simply the power spectral density of $z(r)$. If $z(r)$ is white, then we have Eq. (5.26) with equality, i.e.,

$$\Phi_x(\kappa) = \Phi_z. \tag{5.27}$$

The variance of $x(r)$ is the total power of the $x(r)$ process given by

$$\begin{aligned} \sigma_x^2(r) &= \langle x(r)x^*(r)\rangle \\ &= \begin{cases} \int_{-\infty}^{\infty} d\kappa \, \Phi_x(\kappa), & |r| \le L/2 \\ 0, & \text{else.} \end{cases} \end{aligned} \tag{5.28}$$

Using Eq. (5.24) in Eq. (5.28) yields

$$\begin{aligned} \sigma_x^2(r) &= w(r)R_z(0) \\ &= w(r)\int_{-\infty}^{\infty} d\kappa \Phi_z(\kappa) \\ &= w(r)\sigma_z^2. \end{aligned} \tag{5.29}$$

This result follows directly from Eq. (5.3) and the fact the $z(r)$ is wide-sense-stationary.

5.3. Discrete Case

5.3.1 Autocorrelation Function of Fourier Transform

Let $z(n)$ be a wide-sense-stationary random sequence with zero-mean, autocorrelation sequence $R_z(n)$, and power spectral density $\Phi_z(\nu)$, defined by the equations

$$R_z(n) = \langle z(n' + n)z(n')\rangle, \tag{5.30}$$

$$\Phi_z(\nu) = \sum_n R_z(n)\exp(-2\pi i n\nu). \tag{5.31}$$

We shall let

$$x(n) = w(n)z(n) \tag{5.32}$$

where $w(n)$ is a rectangular window sequence defining the support of $x(n)$.

$$w(n) = \begin{cases} 1, & |n| \leq L/2 \\ 0, & \text{else} \end{cases}. \tag{5.33}$$

The variable ν in $\Phi_z(\nu)$ above is a normalized frequency variable given by

$$\nu = \frac{\kappa}{\kappa_s} \tag{5.34}$$

where κ_s is the sampling frequency. The Fourier transform of $x(n)$ is

$$\begin{aligned} \tilde{x}(\nu) &= \sum_{n=-\infty}^{\infty} x(n) \exp[-2\pi i n \nu] \\ &= \sum_{n=-\infty}^{\infty} w(n)z(n) \exp[-2\pi i n \nu]. \end{aligned} \tag{5.35}$$

Using Eq. (5.35), we form the correlation between two fourier coefficients of $x(n)$:

$$\begin{aligned} \langle \tilde{x}(\nu + \Delta\nu)\tilde{x}^*(\nu) \rangle &= \sum_m \sum_n w(m)w(n) \langle z(m)z^*(n) \rangle \\ &\quad \times \exp[-2\pi i(m-n)\nu - 2\pi i m \Delta\nu] \\ &= \sum_m \sum_n w(m)w(n) R_z(m-n) \exp[-2\pi i(m-n)\nu] \\ &\quad \times \exp[-2\pi i m \Delta\nu]. \end{aligned} \tag{5.36}$$

Now make the following change of variables to Eq. (5.36):

$$k = m - n, \tag{5.37}$$

$$l = m, \tag{5.38}$$

$$n = l - k, \tag{5.39}$$

$$\begin{aligned} \langle \tilde{x}(\nu + \Delta\nu)\tilde{x}^*(\nu) \rangle &= \sum_k \sum_l w(l)w(l-k) R_z(k) \exp[-2\pi i k \nu] \exp[-2\pi i \ell \Delta\nu] \\ &= \sum_k R_z(k) G(k, \Delta\nu) \exp[-2\pi i k \nu], \end{aligned} \tag{5.40}$$

where

$$\begin{aligned} G(k, \Delta\nu) &= \sum_l w(l)w(l-k) \exp[-2\pi i l \Delta\nu] \\ &= \begin{cases} \exp[-\pi i k \Delta\nu] \frac{\sin[\pi\Delta\nu(L+1-|k|)]}{\sin(\pi\Delta\nu)}, & |k| \leq L+1 \\ 0, & |k| > L+1. \end{cases} \end{aligned} \tag{5.41}$$

Substituting Eq. (5.41) into Eq. (5.40), we have

$$\langle \tilde{x}(\nu + \Delta\nu)\tilde{x}^*(\nu) \rangle = \sum_{k=-L-1}^{L+1} R_z(k) \frac{\sin[\pi\Delta\nu(L+1-|k|)]}{\sin(\pi\Delta\nu)} \exp[-2\pi i k (\nu + \Delta\nu/2)]. \tag{5.42}$$

Given any $R_z(k)$, Eq. (5.42) can be used to numerically evaluate the correlation of Fourier coefficients of $x(n)$. However, if we assume that $R_z(k)$ is a "narrow" function compared to the interval $|k| \le L$, i.e.,

$$R_z(k) \simeq 0, \qquad |k| > \ell, \qquad \ell \ll L, \tag{5.43}$$

then Eq. (5.42) can be approximated by

$$\langle \tilde{x}(\nu + \Delta\nu)\tilde{x}^*(\nu)\rangle \simeq \frac{\sin[\pi\Delta\nu(L+1)]}{\sin(\pi\Delta\nu)}\Phi_z(\nu + \Delta\nu/2). \tag{5.44}$$

If $z(n)$ is a white sequence, i.e.,

$$R_z(k) = \Phi_z\delta(k), \tag{5.45}$$

where $\delta(k)$ is the Kronecker delta, then we have Eq. (5.44) with equality,

$$\langle \tilde{x}(\nu + \Delta\nu)\tilde{x}^*(\nu)\rangle = \frac{\sin[\pi\Delta\nu(L+1)]}{\sin(\pi\Delta\nu)}\Phi_z(\nu + \Delta\nu/2). \tag{5.46}$$

We note from consideration of Eq.'s (5.44) and (5.46) that if $R_z(k)$ satisfies the assumption given by Eq. (5.43) the Fourier coefficients of $x(n)$ are uncorrelated for pairs of coefficients associated with frequency differences equal to $\Delta\nu$, when $\Delta\nu$ equals an integer multiple of $1/(L+1)$—a quantity equal to the reciprocal of the support of $x(n)$.

5.3.2 Energy Spectral Density and Power Spectral Density of $x(n)$

The two-sided energy spectral density of any random sequence of finite support is given by

$$E_x(\nu) = \langle|\tilde{x}(\nu)|^2\rangle. \tag{5.47}$$

If $x(n)$ is given by Eq. (5.32), then, from Eq. (5.42),

$$E_x(\nu) = \sum_{k=-L-1}^{L+1} R_z(k)(L+1-|k|)\exp[-2\pi ik\nu]. \tag{5.48}$$

The power spectral density $\Phi_x(\nu)$ of $x(k)$ can be defined as

$$\Phi_x(\nu) = E_x(\nu)/L. \tag{5.49}$$

From Eq. (5.48) we have

$$\Phi_x(\nu) = \frac{1}{L+1}\sum_{k=-L-1}^{L+1} R_z(k)(L+1-|k|)\exp[-2\pi ik\nu]. \tag{5.50}$$

If $R_z(k)$ satisfies the assumption given by Eq. (5.43), then the power spectral density of $x(n)$ is approximated by

$$\Phi_x(\nu) \simeq \Phi_z(\nu). \tag{5.51}$$

If $z(n)$ is white, then we have Eq. (5.51) with equality, i.e.,

$$\Phi_x(\nu) = \Phi_z(\nu). \tag{5.52}$$

The variance of $x(n)$ is the total power of the $x(n)$ process given by

$$\begin{aligned}\sigma_x^2(n) &= \langle|x(n)|^2\rangle \\ &= \begin{cases} \int_{-1/2}^{1/2} d\nu\, \Phi_x(\nu), & |n| \le L/2 \\ 0, & \text{else.}\end{cases}\end{aligned} \tag{5.53}$$

Using Eq. (5.50) in Eq. (5.53) yields

$$\begin{aligned}\sigma_x^2(n) &= w(n)R_z(0) \\ &= w(n)\int_{-1/2}^{1/2} d\nu\, \Phi_z(\nu) \\ &= w(n)\sigma_z^2.\end{aligned} \tag{5.54}$$

This result follows directly from Eq. (5.32) and the fact that $z(n)$ is wide-sense-stationary.

Chapter 6

Object Reconstruction with Sparse Arrays of
Optical Apertures.
Part II:
Nonlinear Methods

(originally issued as TR-1072)

6.1 Introduction

This is the second part of Chapter 4 on the feasibility of using sparse arrays of optical apertures to form useful images of objects. We stress "feasibility," since we have no interest in developing algorithms. In part I[28], we created a discrete one-dimensional model of an optical imaging system, defined a performance measure, and produced results using two linear object reconstruction algorithms, minimum-variance and unweighted least -squares. Using three sparse arrays we concluded that object reconstruction with sparse arrays was indeed feasible so long as object angular support did not exceed the inverse of the size of the gaps between "islands" in the MTF of the array.

Because the algorithms we used were linear, they were capable of producing object reconstructions with negative intensity pixels. In this chapter we explore the question as to whether adding a positivity constraint to an object estimate has the potential of significantly improving performance. We examine two algorithms, unweighted least-squares with a positivity constraint and an algorithm used in radio interferometry called CLEAN. In both cases, we conclude that a positivity constraint has marginal benefit.

Throughout this chapter we assume familiarity with Ref. 28. However, for convenience we have included in their entirety Section 4.2 (optical model) and Section 4.3 (performance measure) of Chapter 4 as Section 6.2 and 6.3 of this report. We have also duplicated Fig.'s 4.1 through 4.4 of Chapter 4 showing three sparse arrays and there MTFs as Fig.'s 6.1 through 6.4 of this chapter.

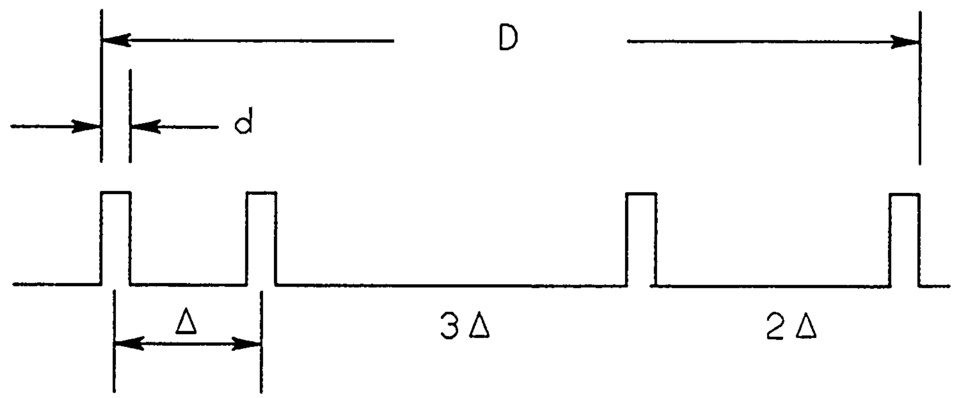
The results for nonlinear algorithms presented in this work were all obtained using Monte Carlo methods, i.e., simulation of a random process, taking averages over many random results. In almost all cases, $N = 40$ random trials were used to provide an average—though in a few cases $N < 40$ random trials were used (if the computational process ran too slowly). In those cases where $N < 40$ was used, we formed an estimate of the variance of the average quantity of interest and used a number of trials, N, large enough so that the accuracy of our estimate of the average quantity of interest would not be seriously compromised.

6.2. Discrete Optical Model (One-Dimensional)

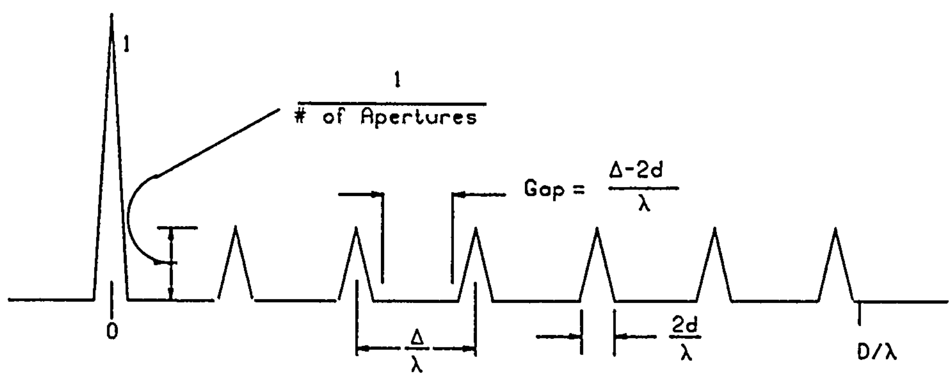
Let the components of the $L \times 1$ vector $\mathbf{z}$ be the intensity pixels of the object line, let the components of the $N \times 1$ vector $\mathbf{y}$ represent the intensity pixels of the image line, and let the $N \times L$ matrix B be the transformation from object line to image line (its columns contain shifted versions of the system point-spread-function). Then an image can be represented by the matrix equation

$$\mathbf{y} = B\mathbf{z} + \mathbf{n}, \tag{6.1}$$

where $\mathbf{n}$ is a $N \times 1$ noise vector. The product $B\mathbf{z}$ in Eq. (6.1) represents convolution of the object line with the point spread-function, followed by truncation of the image. Let the point spread function



a)



b)

Figure 6.1.
Typical sparse array aperture function (a) and the corresponding MTF (b).

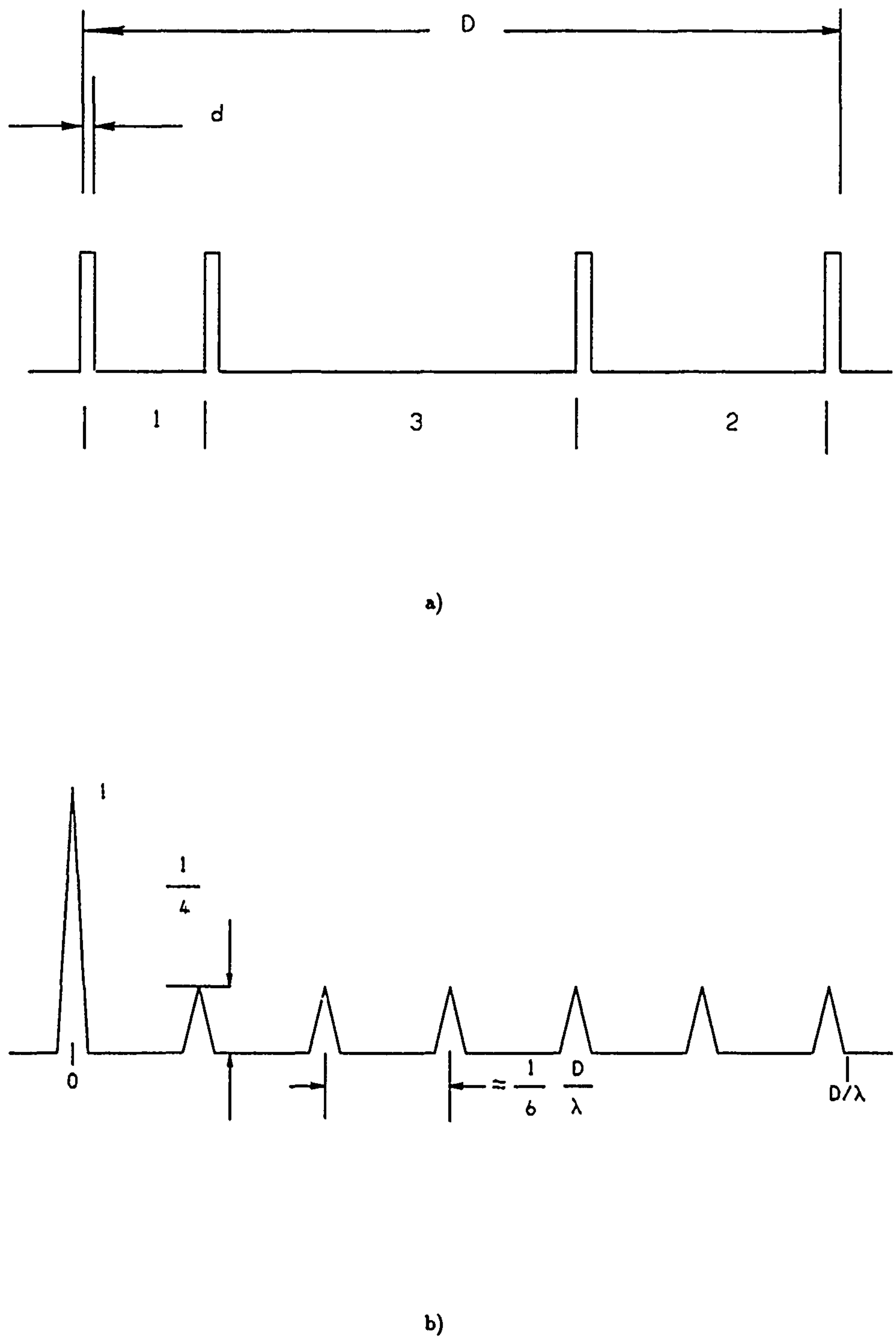
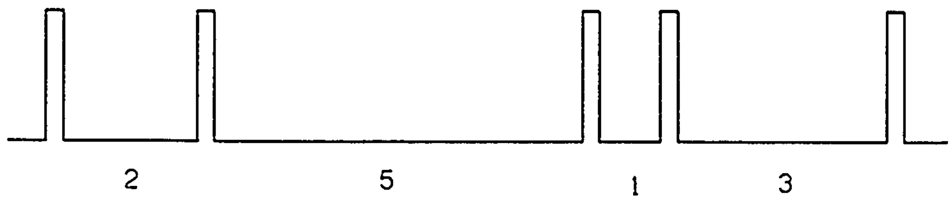
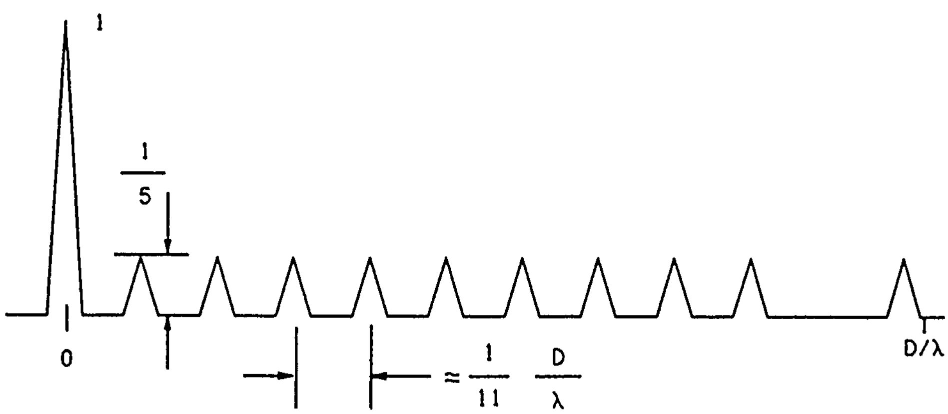


Figure 6.2.
Sparse array aperture function (a) and the corresponding MTF (b) for an "efficient" four-subaperture array.

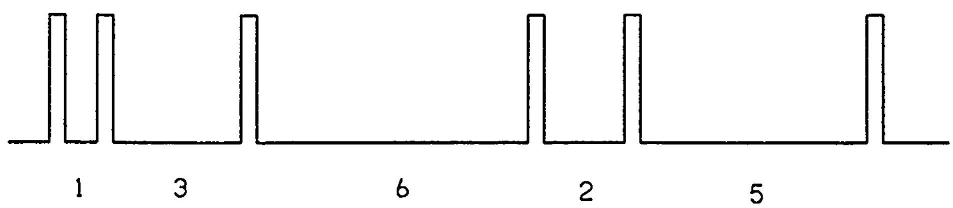


a)

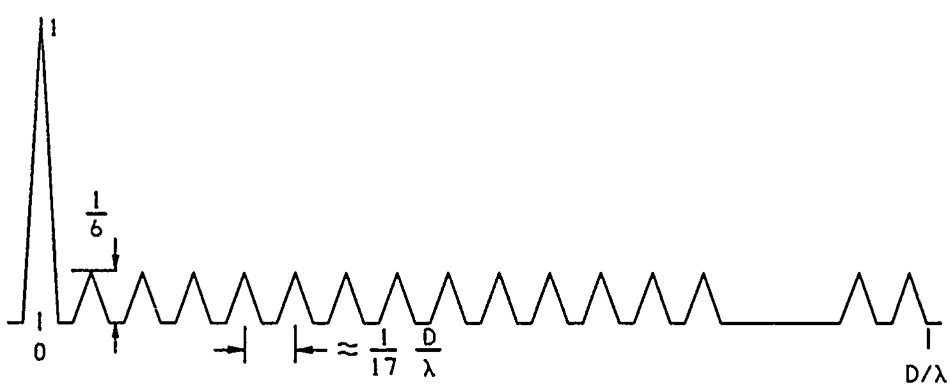


b)

Figure 6.3.
Sparse array aperture function (a) and the corresponding MTF (b) for an "efficient" five-subaperture array.



a)



b)

Figure 6.4.
Sparse array aperture function (a) and the corresponding MTF (b) for an "efficient" six-subaperture array.

be $h(k)$. Then the B matrix is

$$B = \begin{bmatrix}
h(-K) & h(-K-1) & \dots & h(-K-L+1) \\
h(-K+1) & h(-K) & \dots & h(-K-L+2) \\
\vdots & h(-K+1) & \dots & \vdots \\
\vdots & \vdots & \ddots & \vdots \\
\vdots & \vdots & \dots & h(-K-2) \\
h(0) & \vdots & \dots & h(-K-1) \\
\vdots & h(0) & \dots & h(-K) \\
\vdots & \vdots & \dots & h(-K+1) \\
h(K-1) & \vdots & \dots & \vdots \\
h(K) & h(K-1) & \dots & \vdots \\
h(K+1) & h(K) & \dots & h(0) \\
h(K+2) & h(K+1) & \dots & \vdots \\
\vdots & h(K+2) & \dots & \vdots \\
\vdots & \vdots & \dots & h(K-1) \\
h(K+L-1) & h(K+L-2) & \dots & h(K)
\end{bmatrix}, \tag{6.2}$$

where $K = (N-L)/2$.

Now let the $M \times 1$ vector $\mathbf{x}$ represent an object of finite contagious support of length $M \le L$ pixels located somewhere in the object line. We can write

$$\mathbf{z} = W\mathbf{x}, \tag{6.3}$$

where the $L \times M$ matrix W is of the form

$$W = \left[\begin{array}{cccc}
 & & 0 & \\ \hline
1 & & & 0 \\
 & 1 & & \\
 & & \ddots & \\
0 & & & 1 \\ \hline
 & & 0 &
\end{array}\right]. \tag{6.4}$$

Thus, W comprises an $M \times M$ identity matrix embedded in a $L \times M$ matrix of zeros. Combining Eq.'s (6.3) and (6.1) yields

$$\mathbf{y} = BW\mathbf{x} + \mathbf{n}. \tag{6.5}$$

To complete the model, we need a point-spread function. Let $w(x)$ be an arbitrary aperture function. The corresponding modulation transfer function (MTF) is given below, where κ is the spatial frequency variable in units of cycles per radian and λ is wavelength. We write

$$MTF(\kappa) = \frac{1}{A}\int dx\, w(x + \tfrac{1}{2}\kappa\lambda)w(x - \tfrac{1}{2}\kappa\lambda) \tag{6.6}$$

A is chosen to be the area (in this case the length) of the aperture, so that the MTF at zero spatial frequency is unity.

The point-spread function of an optical system is the inverse Fourier transform of the MTF function, or

$$h_c(x) = \int d\kappa\, \mathrm{MTF}(\kappa) \exp[2\pi i \kappa x]. \tag{6.7}$$

The subscript c denotes that this point-spread function is the continuous version. The discrete version of the point-spread function is found by sampling and scaling the continuous version. Let δ denote the pixel spacing in the image line. Then the discrete point spread function used in the B matrix given by Eq. (6.2) is

$$h(n) = \delta h_c(n\delta); \quad n = 0, \pm 1, \pm 2, \cdots. \tag{6.8}$$

The continuous point spread functions for the four, five, and six subaperture arrays are given in Eq.'s (9), (10), and (11) respectively. (The reader should see Fig.s 6.2, 6.3, and 6.4 for the corresponding MTF's.) We have

$$h_c(x) = \frac{d}{\lambda}\left(\frac{\sin \pi \frac{d}{\lambda} x}{\pi \frac{d}{\lambda} x}\right)^2 \left(1 + \frac{1}{2}\sum_{i=1}^{6} \cos\left[i \frac{\pi}{3}\frac{D}{\lambda}\left(1 - \frac{d}{D}\right) x\right]\right), \tag{6.9}$$

$$h_c(x) = \frac{d}{\lambda}\left(\frac{\sin \pi \frac{d}{\lambda} x}{\pi \frac{d}{\lambda} x}\right)^2 \left(1 + \frac{2}{5}\sum_{\substack{i=1 \\ i\neq 10}}^{11} \cos\left[i \frac{2\pi}{11}\frac{D}{\lambda}\left(1 - \frac{d}{D}\right) x\right]\right), \tag{6.10}$$

$$h_c(x) = \frac{d}{\lambda}\left(\frac{\sin \pi \frac{d}{\lambda} x}{\pi \frac{d}{\lambda} x}\right)^2 \left(1 + \frac{1}{3}\sum_{\substack{i=1 \\ i\neq 14 \\ i\neq 15}}^{17} \cos\left[i \frac{2\pi}{17}\frac{D}{\lambda}\left(1 - \frac{d}{D}\right) x\right]\right). \tag{6.11}$$

6.3. A Performance Measure

In the sections to follow we investigate two algorithms for object recovery. In order to evaluate the performance of each algorithm and to compare algorithms, we need a measure of performance that we can apply uniformly to both algorithms. Our object recovery algorithms not only attempt to fill in the gaps between measured spatial frequency components of the object (interpolative super-resolution), but also to estimate the object at spatial frequencies beyond D/λ (extrapolative super-resolution), where D is the over all length of the array. Because of array geometries, our algorithms will be performing better at recovering some spatial frequency components than others. We therefore need a frequency sensitive performance measure. To this end, let $x(n)$ be the n^{th} pixel of the object, $\widehat{x}(n)$ be the n^{th} pixel of an estimate of the object, and $e(n)$ be the n^{th} pixel of the estimation error given by

$$e(n) = x(n) - \widehat{x}(n), \tag{6.12a}$$

or in vector notation,

$$\mathbf{e} = \mathbf{x} - \widehat{\mathbf{x}}. \tag{6.12b}$$

One common measure of performance is mean-square-error, $\langle \mathbf{e}^T \mathbf{e} \rangle$. For our purposes this measure is not useful. It gives us no indication of how well an algorithm performs as a function of spatial frequency. Instead, the performance measure we will use is a signal-to-noise ratio (SNR) that is a function of a spatial cutoff frequency. The behavior of the SNR as a function of cutoff frequency will be an indication of the spatial resolution of the sparse array. To introduce frequency sensitivity into the performance measure, let $x_{\kappa_0}(n)$, $\widehat{x}_{\kappa_0}(n)$, and $e_{\kappa_0}(n)$ be filtered versions of $x(n)$, $\widehat{x}(n)$, and

$e(n)$, respectively, where the filtering is ideal low-pass with cutoff frequency κ_o. To explicitly define the filtering operation, let $y(n)$ be a sequence of pixels and $\tilde{y}(\kappa)$ be its Fourier transform given by

$$\tilde{y}(\kappa) = \sum_n y(n) \exp(-2\pi i n \kappa \delta), \tag{6.13}$$

where δ is the object pixel spacing. Let $\tilde{f}(\kappa)$ be the filter transfer function given by

$$\tilde{f}(\kappa) = \begin{cases} 1, & |\kappa| \le \kappa_o \\ 0, & |\kappa| > \kappa_o. \end{cases} \tag{6.14}$$

Then the Fourier transform of $y_{\kappa_o}(n)$, the filtered version of $y(n)$, is given by

$$\tilde{y}_{\kappa_o}(\kappa) = \tilde{y}(\kappa)\tilde{f}(\kappa). \tag{6.15}$$

Thus, we have

$$\tilde{x}_{\kappa_o}(\kappa) = \tilde{x}(\kappa)\tilde{f}(\kappa) \tag{6.16}$$

$$\tilde{\hat{x}}_{\kappa_o}(\kappa) = \tilde{\hat{x}}(\kappa)\tilde{f}(\kappa) \tag{6.17}$$

$$\begin{aligned} \tilde{e}_{\kappa_o}(\kappa) &= \tilde{e}(\kappa)\tilde{f}(\kappa) \\ &= \left[\tilde{x}(\kappa) - \tilde{\hat{x}}(\kappa)\right] \tilde{f}(\kappa) \\ &= \tilde{x}_{\kappa_o}(\kappa) - \tilde{\hat{x}}_{\kappa_o}(\kappa). \end{aligned} \tag{6.18}$$

Let us now define a signal-to-noise ratio for the filtered estimate $\hat{x}_{\kappa_o}(n)$ as the ratio of the average energy of the filtered object to the average energy of the filtered estimation error:

$$\mathrm{SNR_{EST}}(\kappa_o) = \frac{\left\langle \sum_n x^2_{\kappa_o}(n) \right\rangle}{\left\langle \sum_n e^2_{\kappa_o}(n) \right\rangle}. \tag{6.19}$$

The angle brackets in Eq. (6.19) denote ensemble average. From Parseval's theorem we know that the quantities inside the angle brackets in Eq. (6.19) can be written in terms of their Fourier transforms, as follows:

$$\sum_n x^2_{\kappa_o}(n) = \delta \int_{-1/2\delta}^{1/2\delta} d\kappa\, |\tilde{x}_{\kappa_o}(\kappa)|^2 \tag{6.20}$$

$$\sum_n e^2_{\kappa_o}(n) = \delta \int_{-1/2\delta}^{1/2\delta} d\kappa\, |\tilde{e}_{\kappa_o}(\kappa)|^2. \tag{6.21}$$

We can use Eq.'s (6.13), (6.14), (6.16), and (6.18) to rewrite Eq.'s (6.20) and (6.21), yielding

$$\begin{aligned} \sum_n x^2_{\kappa_o}(n) &= \delta \int_{-1/2\delta}^{1/2\delta} d\kappa\, |\tilde{x}(\kappa)|^2 |\tilde{f}(\kappa)|^2 \\ &= 2\delta \int_0^{\kappa_o} d\kappa\, |\tilde{x}(\kappa)|^2 \end{aligned} \tag{6.22}$$

$$\begin{aligned} \sum_n e^2_{\kappa_o}(n) &= \delta \int_{-1/2\delta}^{1/2\delta} d\kappa\, |\tilde{e}(\kappa)|^2 |\tilde{f}(\kappa)|^2 \\ &= 2\delta \int_0^{\kappa_o} d\kappa\, |\tilde{e}(\kappa)|^2. \end{aligned} \tag{6.23}$$

Replacing the quantities inside the angle-brackets in Eq. (6.19) with their corresponding expressions given by Eq.'s (6.22) and (6.23), and bringing the angle-brackets inside the integrals, yields

$$\mathrm{SNR_{EST}}(\kappa_o) = \frac{\int_0^{\kappa_o} \mathrm{d}\kappa \,\langle|\widetilde{\mathrm{x}}\kappa|^2\rangle}{\int_0^{\kappa_o} \mathrm{d}\kappa\langle|\widetilde{\mathrm{e}}(\kappa)|^2\rangle}. \tag{6.24}$$

Since integrating the quantities $\delta\langle|\widetilde{x}(\kappa)|^2\rangle$ and $\delta\langle|\widetilde{e}(\kappa)|^2\rangle$ over the spatial frequency band $|\kappa| \leq \kappa_o$ yields, respectively, the energy in the object and estimation error over the same band, we will call these quantities energy spectral densities or energy spectra, and use the notation

$$E_x(\kappa) = \delta\langle|\widetilde{x}(\kappa)|^2\rangle \tag{6.25}$$

$$E_e(\kappa) = \delta\langle|\widetilde{e}(\kappa)|^2\rangle. \tag{6.26}$$

So that

$$\mathrm{SNR_{EST}}(\kappa_o) = \frac{\int_0^{\kappa_o} \mathrm{d}\kappa \,\mathrm{E_x}(\kappa)}{\int_0^{\kappa_o} \mathrm{d}\kappa \,\mathrm{E_e}(\kappa)}. \tag{6.27}$$

To compute the energy spectrum of a random sequence, we simply follow the prescription given by Eq.'s (6.25) and (6.13). Let $\mathbf{z}$ be a random vector with n^{th} component $z(n)$. Then its energy spectrum is given by

$$\begin{aligned} E_z(\kappa) &= \delta\langle|\widetilde{z}(\kappa)|^2\rangle \\ &= \delta\left\langle\left|\sum_n z(n)\exp(-2\pi i n\kappa\delta)\right|^2\right\rangle \\ &= \delta\sum_n\sum_m\langle z(n)z(m)\rangle\exp[-2\pi i(n-m)\kappa\delta]. \end{aligned} \tag{6.28}$$

We note from Eq. (6.28) that we need *all* entries of the covariance matrix of $z(n)$ in the evaluation of its energy spectrum.

6.4. Unweighted Least-Squares, Finite Support and Positivity Constraints

In Section 6.5. of Ref. 28 we used unweighted least-squares for object reconstruction, i.e., given the observation

$$\mathbf{y} = G\mathbf{x} + \mathbf{n}, \tag{6.29}$$

pick the object vector estimate $\widehat{\mathbf{x}}$ which minimizes

$$\epsilon = ||\mathbf{y} - G\widehat{\mathbf{x}}||^2, \tag{6.30}$$

where $\mathbf{x}$ is the object vector, $\mathbf{n}$ is observation noise, and the columns of G are the system point-spread-function shifted and truncated. The finite support constraint to the solution is implicitly contained in the dimension of the object vector $\mathbf{x}$ and its estimate $\widehat{\mathbf{x}}$. To add the positivity constraint we simply add the statement

$$\widehat{\mathbf{x}} \geq 0. \tag{6.31}$$

The object vector estimate that minimizes ϵ of Eq. (6.30) can, of course, be obtained in closed form. Unfortunately, this is not the case when the positivity constraint given by Eq. (6.31) is included. One must use some numerical iteration method for each case of an object vector and a noise vector, and perform some averaging of the results. Therefore, to evaluate the error vector covariance matrix when using positivity as a constraint requires a simulation. To generate object vectors, we generated independent Rayleigh random variables for each pixel. Thus, object intensity

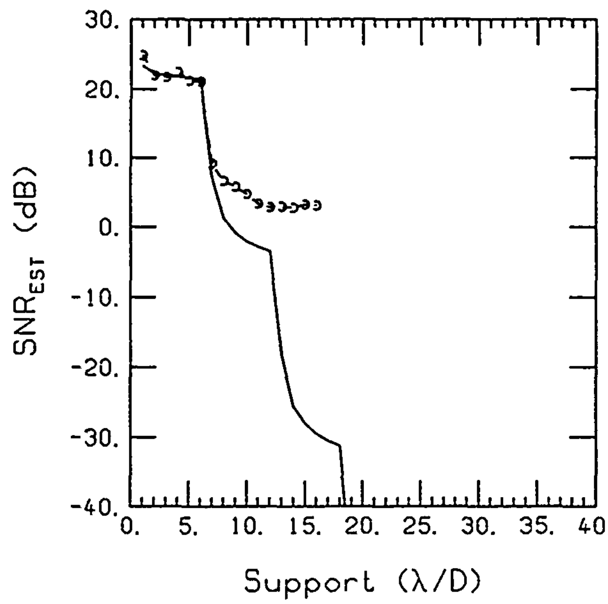


Figure 6.5.
Signal-to-noise ratio of the object estimate versus both assumed and actual object support using the unweighted least-square processor without and with a positivity constraint on the object estimate (solid line and circles, respectively). The four-subaperture array of Fig. 2 was used with $SNR_{REF} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 1.0D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, **y**, used here had a length of 256 λ/D.

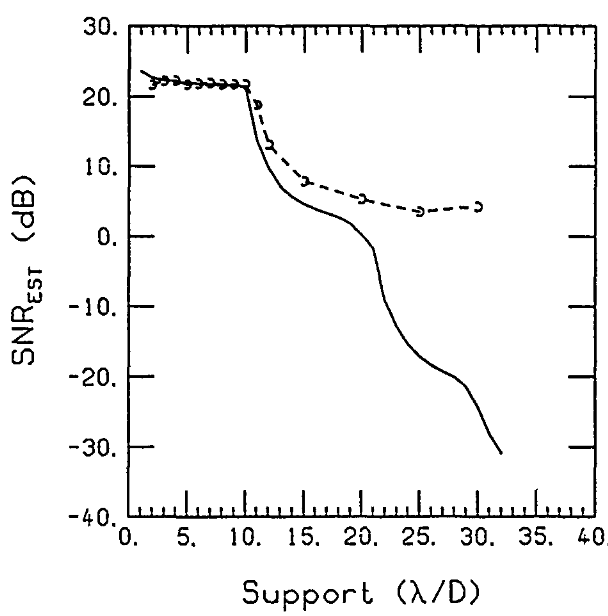


Figure 6.6
Signal-to-noise ratio of the object estimate versus both assumed and actual object support using the unweighted least-squares processor without and with a positivity constraint on the object estimate (solid line and circles, respectively). The five-subaperture array of Fig. 3 was used with $SNR_{REF} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 0.81D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, **y**, used here had a length of 256 λ/D.

pixels were always positive. Noise vectors were generated using independent, zero-mean, Gaussian random variables. An image vector, $\mathbf{y}$, was generated using Eq. (6.29) for each sample object and noise vector generated. The object vector estimate, $\hat{\mathbf{x}}$, for each case was found using an algorithm called gradient projection[2]. It is essentially the steepest descent algorithm modified to accommodate the positivity constraint on the estimate. The algorithm is described in detail in Appendix A. If we let $\mathbf{x}(n)$ be the object vector and $\hat{\mathbf{x}}(n)$ be the object vector estimate, each for the n^{th} random trial, then the error covariance matrix was estimated using

$$\widehat{\langle \mathbf{e}\mathbf{e}^T \rangle} = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}(n) - \hat{\mathbf{x}}(n))\,(\hat{\mathbf{x}}(n) - \hat{\mathbf{x}}(n))^T\,, \tag{6.32}$$

where N denotes the total number of random trials. The energy spectrum of the error vector was computed using Eq. (6.32) in Eq. (6.28). The results were then integrated to compute $\mathrm{SNR_{EST}}$, as was done in Ref. 28. Plots of $\mathrm{SNR_{EST}}$ versus object support are shown in Fig.'s 6.5 through 6.7 for the four, five, and six subaperture cases, respectively. These calculations were performed with $N = 40$. Note that the use of a positivity constraint tends to moderate the estimate in those cases where the SNR is low, just as the use of statistical knowledge in the minimum-variance processor did. However, just as in the minimum-variance case, positivity appears to be of little help in performance improvement at SNR levels that are probably the minimum useful levels.* We also performed simulations at $\mathrm{SNR_{REF}}$ levels higher and lower than the 50 dB used to generate the results of Fig.'s 6.5 through 6.7. However, there were no surprises in the results. At $\mathrm{SNR_{REF}} > 50$ dB the performance in the region where support is less than the reciprocal of the gap in the MTF "island" was the same with and without the positivity constraint. At $\mathrm{SNR_{REF}} < 50$ dB the performance with positivity was marginally better than the performance without positivity in the same support region, with the performance gap increasing as $\mathrm{SNR_{REF}}$ is lowered. However, it is questionable whether the performance improvement with positivity is useful at such low levels of $\mathrm{SNR_{EST}}$.

The above results were obtained with assumed and actual object support the same. The question arose as to the result if assumed object support were larger than actual object support, i.e., the object estimate the processor is allowed to formulate has a larger support than the actual size of the object. Until we introduced the positivity constraint, this question had no meaning. The minimum-variance processor requires second moment information about the object, and thus its size. Therefore, assumed and actual object size cannot be different. For unweighted least-squares without positivity, it can be shown that performance depends only on assumed object support, and as we have shown in Ref. 1, this must be less than the reciprocal of the gap in the MTF "islands" for good performance. However, for the case of unweighted least-squares with positivity, the results were somewhat surprising. We found that the critical size is neither the assumed nor actual object support, but their average, i.e., if the average of the assumed and actual object supports is smaller than the "gap", then $\mathrm{SNR_{EST}}$ will not fall off the "cliff". This result is illustrated in Fig.'s 6.8 through 6.10, corresponding to the four, five, and six subaperture arrays respectively. The same results are shown in Fig.'s 6.11 through 6.13 with an expanded vertical scale to show more detail. (The simulation results shown in Fig.'s 6.8, 6.9, 6.11, and 6.12 used $N = 40$ random trials, and the simultion results shown in Fig.'s 6.10 and 6.13 used $N = 20$ random trials.) Although this feature of positivity is interesting, it is hard to imagine a situation in which it would be useful.

6.5 CLEAN

The computational complexity of unweighted least-squares with a positivity constraint compels one to find a less burdensome algorithm that also incorporates a positivity constraint on the object estimate. One such algorithm is CLEAN[2], used in radio interferometry to reconstruct objects from

* Note that an $\mathrm{SNR_{EST}}$ of 20 dB corresponds to a "voltage" ratio of 10.

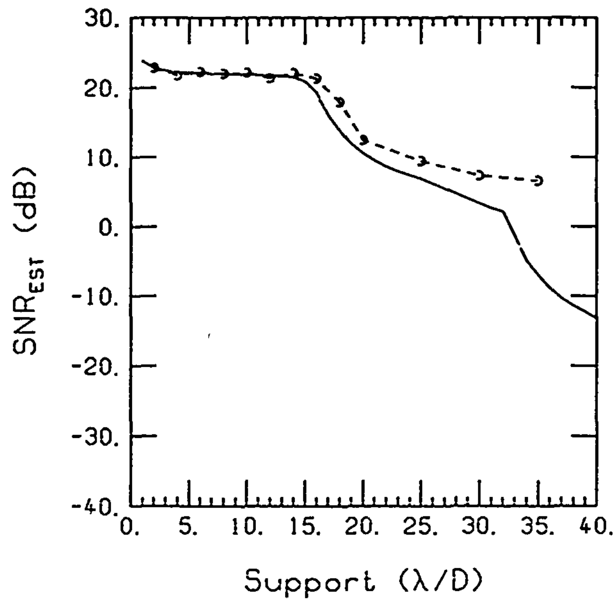


Figure 6.7.

Signal-to-noise ratio of the object estimate versus both assumed and actual object support using the unweighted least-square processor without and with a positivity constraint on the object estimate (solid line and circles, respectively). The six-subaperture array of Fig. 6.4 was used with $\mathrm{SNR_{REF}} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 0.76D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, $\mathbf{y}$, used here had a length of 256 λ/D.

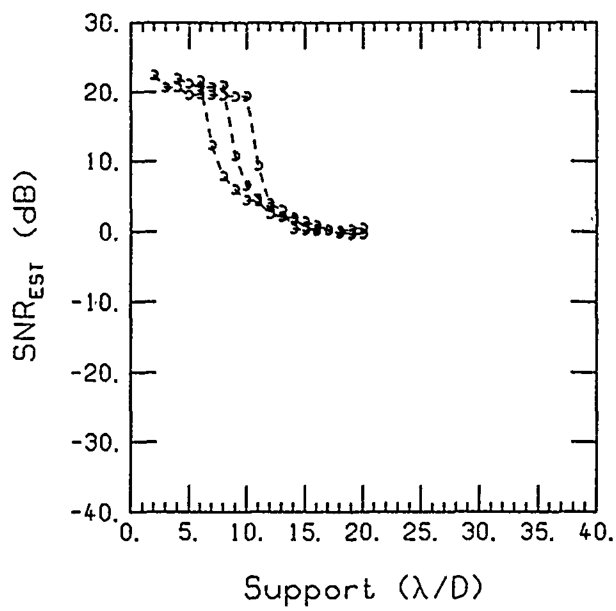


Figure 6.8.

Signal-to-noise ratio of the object estimate versus assumed object support using the unweighted least-squares processor with a positivity constraint on the object estimate. The four-subaperture array of Fig. 6.2 was used with $\mathrm{SNR_{REF}} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 1.0D/\lambda$. From right to left, the three curves correspond to actual object sizes of 2, 4, and 6 λ/D. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, $\mathbf{y}$, used here had a length of 256 λ/D.

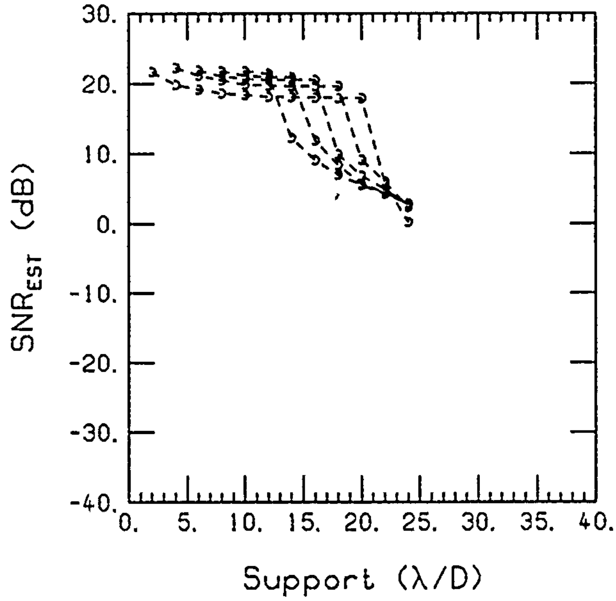


Figure 6.9.

Signal-to-noise ratio of the object estimate versus assumed object support using the unweighted least-squares processor with a positivity constraint on the object estimate. The five-subaperture array of Fig. 6.3 was used with $SNR_{REF} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 0.81D/\lambda$. From right to left, the five curves correspond to actual object sizes of 2, 4, 6, 8, and 10 λ/D. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, **y**, used here had a length of 256 λ/D.

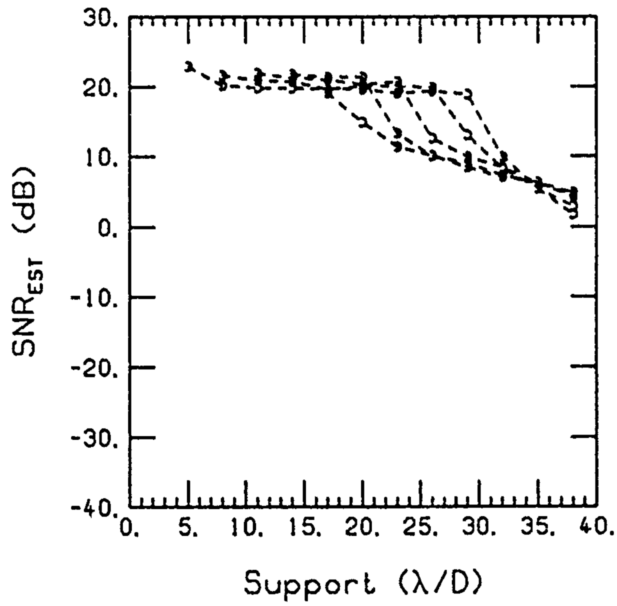


Figure 6.10.

Signal-to-noise ratio of the object estimate versus assumed object support using the unweighted least-squares processor with a positivity constraint on the object estimate. The six-subaperture array of Fig. 6.4 was used with $SNR_{REF} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 0.76D/\lambda$. From right to left, the five curves correspond to actual object sizes of 5, 8, 11, 14, and 17 λ/D. There were $N = 20$ Monte Carlo runs used to generate the simulation data shown here. The image data array, **y**, used here had a length of 256 λ/D.

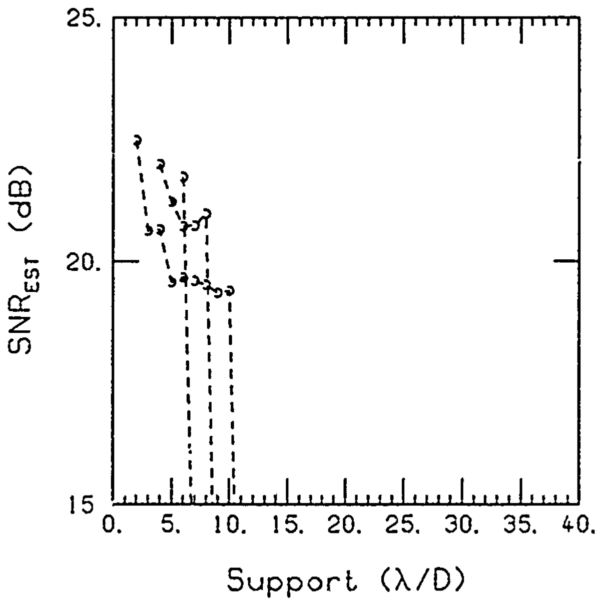


Figure 6.11.

Signal-to-noise ratio of the object estimate versus assumed object support using the unweighted least-squares processor with a positivity constraint on the object estimate. The four-subaperture array of Fig. 6.2 was used with $SNR_{REF} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 1.0D/\lambda$. From right to left, the three curves correspond to actual object sizes of 2, 4, and 6 λ/D. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, $\mathbf{y}$, used here had a length of 256 λ/D.

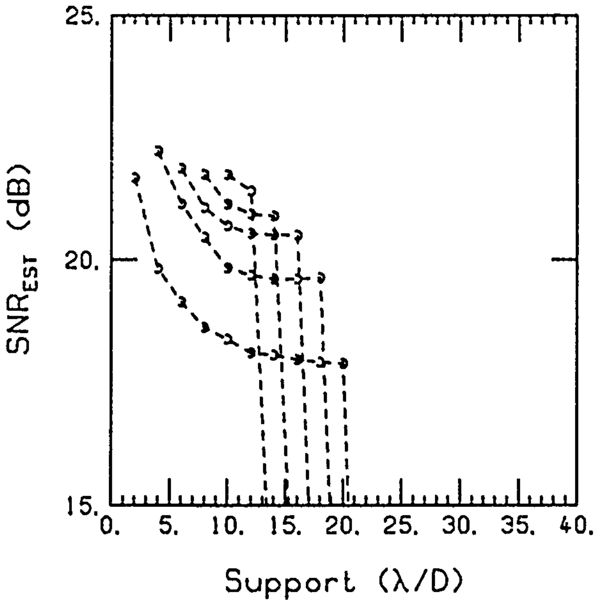


Figure 6.12.

Signal-to-noise ratio of the object estimate versus assumed object support using the unweighted least-squares processor with a positivity constraint on the object estimate. The five-subaperture array of Fig. 6.3 was used with $SNR_{REF} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 0.81D/\lambda$. From right to left, the five curves correspond to actual object sizes of 2, 4, 6, 8, and 10 λ/D. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, $\mathbf{y}$, used here had a length of 256 λ/D.

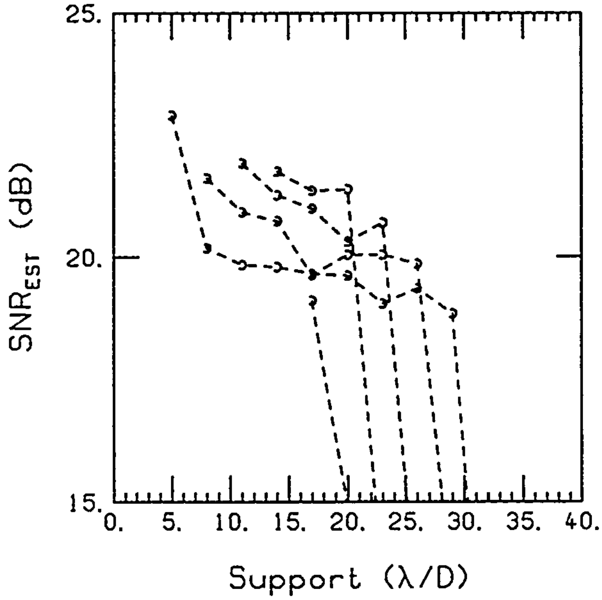


Figure 6.13.

Signal-to-noise ratio of the object estimate versus assumed object support using the unweighted least-squares processor with a positivity constraint on the object estimate. The six-subaperture array of Fig. 6.4 was used with $SNR_{REF} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 0.76D/\lambda$. From right to left, the five curves correspond to actual object sizes of 5, 8, 11, 14, and 17 λ/D. There were $N = 20$ Monte Carlo runs used to generate the simulation data shown here. The image data array, $\mathbf{y}$, used here had a length of 256 λ/D.

interferometric data. Since radio interferometers certainly employ sparse arrays, we were curious as to whether CLEAN was subject to the same limitations vis-a-vis object support and MTF gaps as the other algorithms that we have tried. CLEAN begins with the same observation model that we have used throughout our investigation:

$$\mathbf{y} = G\mathbf{x} + \mathbf{n}, \tag{6.33}$$

where $\mathbf{x}$ is the object vector, $\mathbf{n}$ is observation noise, and the columns of G are the system point-spread-function shifted and truncated. CLEAN is an iterative attempt to solve the system of equations

$$\mathbf{y} = G\hat{\mathbf{x}}, \tag{6.34}$$

with the positivity constraint

$$\hat{\mathbf{x}} \geq 0, \tag{6.35}$$

where $\mathbf{y}$ and G are given, and $\hat{\mathbf{x}}$ is unknown. If the solution exists (in our case that is equivalent to no noise), and under the appropriate circumstances, CLEAN is capable of converging to the exact solution. However, if there is observation noise, then CLEAN will not in general converge to the best solution in the constrained least-squares sense, i.e., CLEAN will not minimize ϵ of Eq. (6.30) with the constraint given by Eq. (6.31). Furthermore, as we shall see, CLEAN does not in general perform as well in terms of SNR_{EST} as least-squares. Details of our implementation of CLEAN are contained in Appendix E.

Simulations using CLEAN were carried out in the same way as for unweighted least-squares with positivity. Independent Rayleigh and Gaussian random variables were generated to create object and noise vectors, the image vector $\mathbf{y}$ was computed, and CLEAN was applied to form an estimate of the object vector. After many trials (in all cases presented in this report, we used $N = 40$ random trials), a sample error covariance matrix was computed using Eq. (6.32).

SNR_{EST} versus object support is plotted in Fig.'s 6.14, 6.15, and 6.16, for SNR_{REF} = 50 dB, 60 dB, and 70 dB, respectively. The circles represent the results of simulations using CLEAN and the solid curves are the results using unweighted least-squares. In all cases, the four-subaperture nonredundant array of Fig. 6.2 was used. The first characteristic to note in all three figures is that CLEAN exhibits the same "cliff" behavior as the other algorithms we have tried; object support must be smaller than the reciprocal of the gaps between the "islands" in the MTF or SNR_{EST} falls off the "cliff". Also, the positivity constraint intrinsic in CLEAN moderates the object estimate at low SNR, just as does positivity applied to unweighted least-squares or statistical knowledge in the minimum-variance processor. Finally, we note that, to the left of the "cliff", that CLEAN tracks least- squares with about a 14 dB performance loss. We will explore the reason for this loss shortly. However, we will now look at the CLEAN results using the other two nonredundant arrays shown in Fig.'s 6.3 and 6.4 and discover a limitation of CLEAN not suffered by the other algorithms we have tried. We begin with the 5-subaperture array of Fig. 6.3 and the corresponding CLEAN results of Fig. 6.17 with SNR_{REF} = 50 dB. There are no surprises in this figure. The results are what we would expect in the light of previous results. Now consider the results using SNR_{REF} = 60 dB shown in Fig. 6.18. We now see a considerable falloff in CLEAN performance beyond an object support of $5\lambda/D$. The result is more exaggerated in Fig. 6.19 were SNR_{REF} = 70 dB. The "cliff" appears to be located at $5\lambda/D$ instead of the expected location of 10 to 11 λ/D. For the present, we simply point out that the MTF of the five-subaperture array has a missing "island", so that one of the gaps is $2/11D/\lambda$, the reciprocal of which is 5.5 λ/D. Now consider the results shown in Fig.'s 6.20 through 6.22 using the six-subaperture array of Fig. 6.4. Again we have a severe loss in performance for object support exceeding 5 λ/D when we would expect level performance out to a support of 16 to 17 λ/D. And again, we note that the six-subaperture array has two adjacent missing "islands" in its MTF with a corresponding gap of $3/17D/\lambda$, the reciprocal of which is 5.67 λ/D. Apparently, CLEAN is confused by irregularly spaced "islands" in the MTF of the array. In other words, CLEAN appears to require that object support be less than the reciprocal of the largest gap in the MTF of the array, even though that gap is outside the cutoff frequency used in computing SNR_{EST}. We gain additional insight by examining the energy spectrum of the error vector when using CLEAN. Fig. 6.23 contains curves of energy spectrum versus spatial frequency using the five-subaperture array. Note that CLEAN is successfully interpolating between "islands" in the MTF for objects with support less than 12 λ/D, but the performance gradually degrades beyond a support of 5 λ/D. Fig. 6.24 reveals the same behavior using the six-subaperture array. The obvious solution is to spatially filter the observation vector $\mathbf{y}$ with a low-pass filter with a cutoff frequency equal to the upper bound of the "useful range" of the array. The point-spread function used in the CLEAN algorithm would be adjusted to reflect the "new" MTF. To demonstrate the concept while avoiding the filtering operation, we instead created two redundant arrays with the same "island" spacing as the nonredundant five and six subaperture arrays. These arrays and there corresponding MTFs are shown in Fig.'s 6.25 and 6.26, respectively, and the results of the simulations are shown in Fig.'s 6.27 through 6.32. Note that in every case we now have the behavior that we originally expected with the nonredundant arrays.

We now refer back to one of the early observations, that the performance of CLEAN is worse than least-squares in the region where object support is less than the reciprocal of the "gap" size in the MTF. In an optical system using a completely filled (nonsparse) aperture, there is very little information in the image plane about the object being imaged outside the support region of the object. This not the case with a very sparse array. As the "sparseness" is increased, information about the object is spread further out into the image plane, considerable beyond the region of object support. All of this information is potentially useful in a signal-to-noise ratio sense, and the algorithm that can use it will perform better than one that cannot. However, there is usually a point of diminishing returns, where increasing the size of the observation region will no longer significantly improve performance because of the increased amount of observation noise. In all the results that we have presented, both in this report and in Ref. 28, using minimum-variance and least-squares

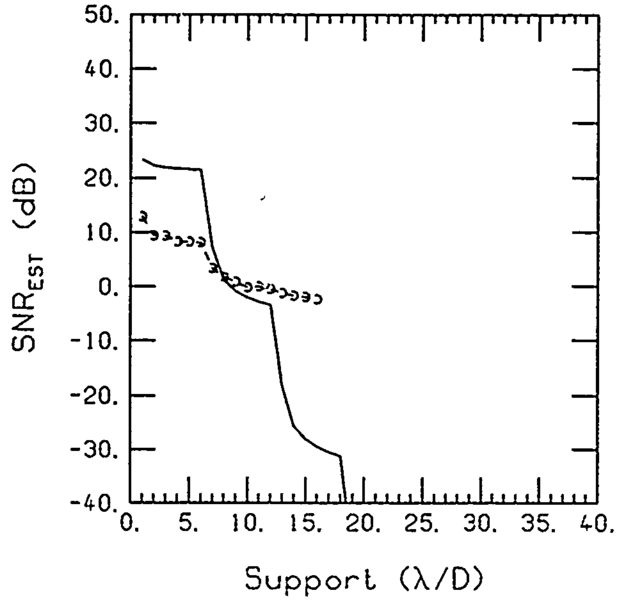


Figure 6.14.
Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The four-subaperture array of Fig. 6.2 was used with $\mathrm{SNR_{REF}} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 1.0D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, **y**, used here had a length of 256 λ/D.

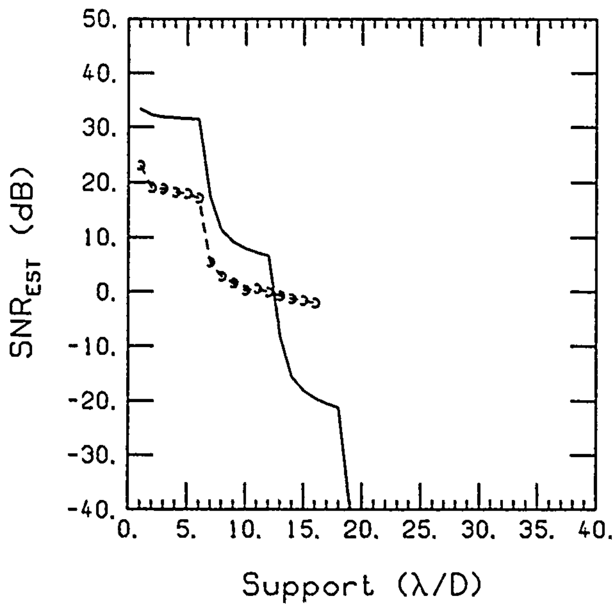


Figure 6.15.
Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The four-subaperture array of Fig. 6.2 was used with $\mathrm{SNR_{REF}} = 60$ dB, $d/D = 0.005$, and $\kappa_0 = 1.0D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, **y**, used here had a length of 256 λ/D.

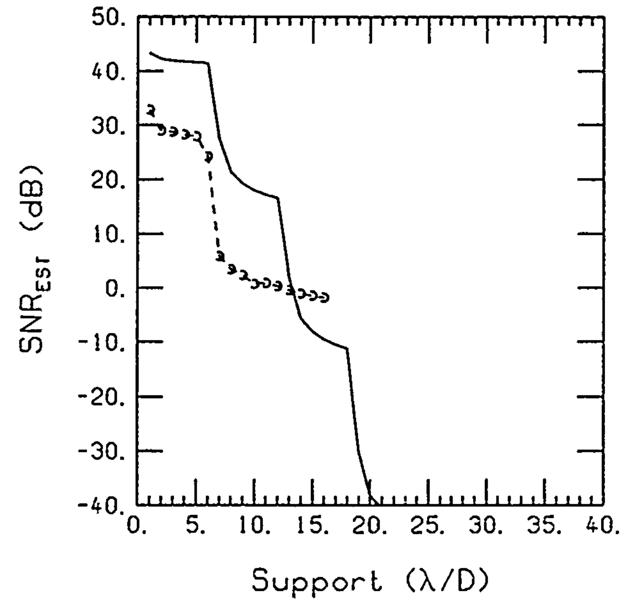


Figure 6.16.
Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The four-subaperture array of Fig. 6.2 was used with $\mathrm{SNR_{REF}} = 70$ dB, $d/D = 0.005$, and $\kappa_0 = 1.0D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, **y**, used here had a length of 256 λ/D.

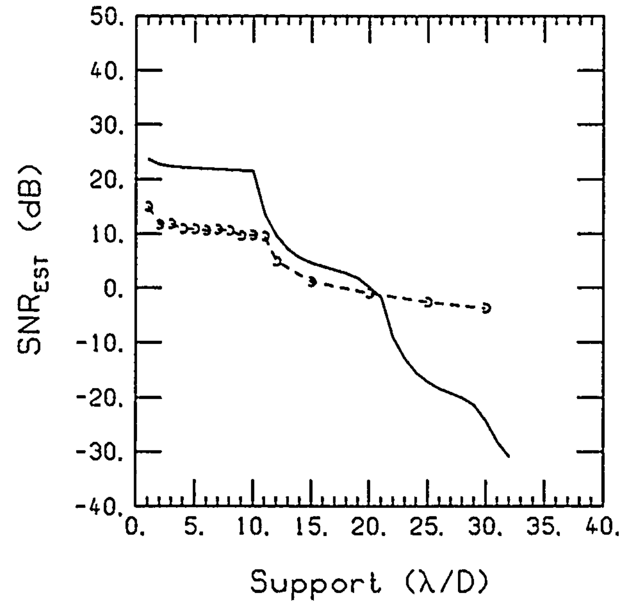


Figure 6.17.
Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The five-subaperture array of Fig. 6.3 was used with $\mathrm{SNR_{REF}} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 0.81D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, **y**, used here had a length of 256 λ/D.

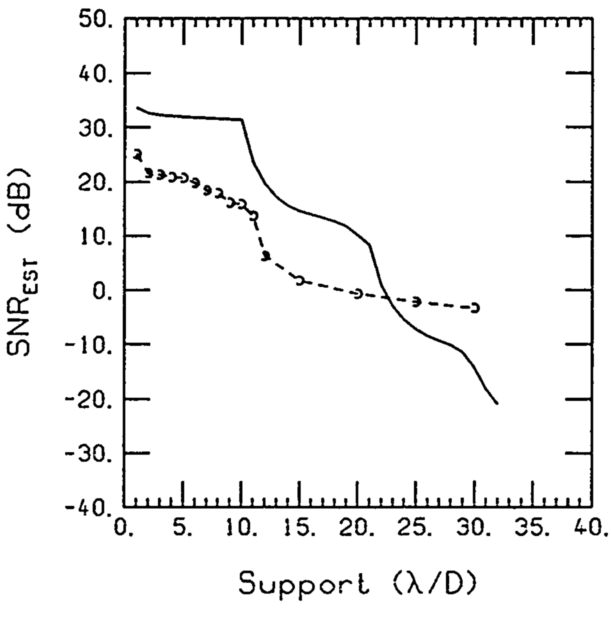


Figure 6.18.
Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The five-subaperture array of Fig. 6.3 was used with $\text{SNR}_{\text{REF}} = 60$ dB, $d/D = 0.005$, and $\kappa_0 = 0.81D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, $\mathbf{y}$, used here had a length of 256 λ/D.

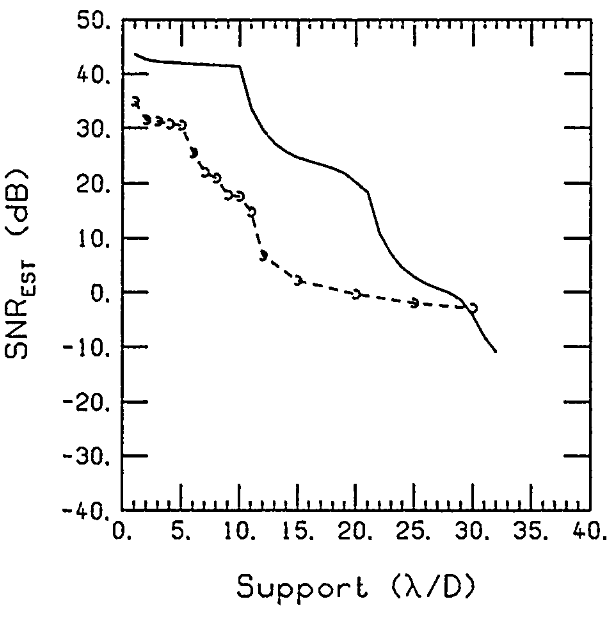


Figure 6.19.
Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The five-subaperture array of Fig. 6.3 was used with $\text{SNR}_{\text{REF}} = 70$ dB, $d/D = 0.005$, and $\kappa_0 = 0.81D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, $\mathbf{y}$, used here had a length of 256 λ/D.

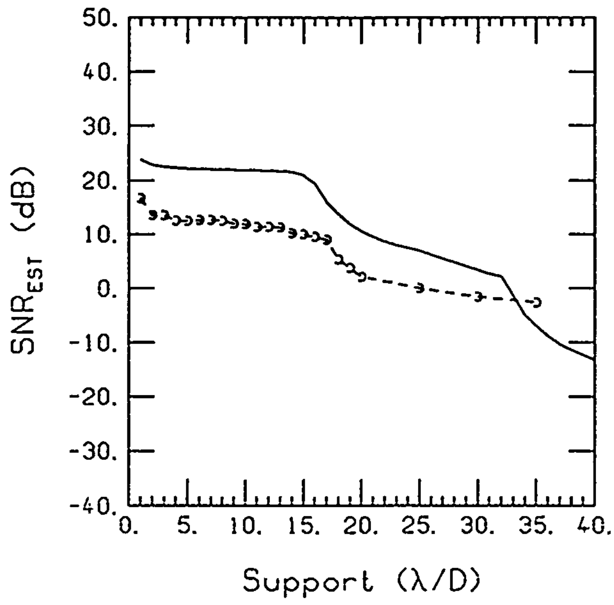


Figure 6.20.
Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The six-subaperture array of Fig. 6.4 was used with $SNR_{REF} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 0.76D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, $\mathbf{y}$, used here had a length of 256 λ/D.

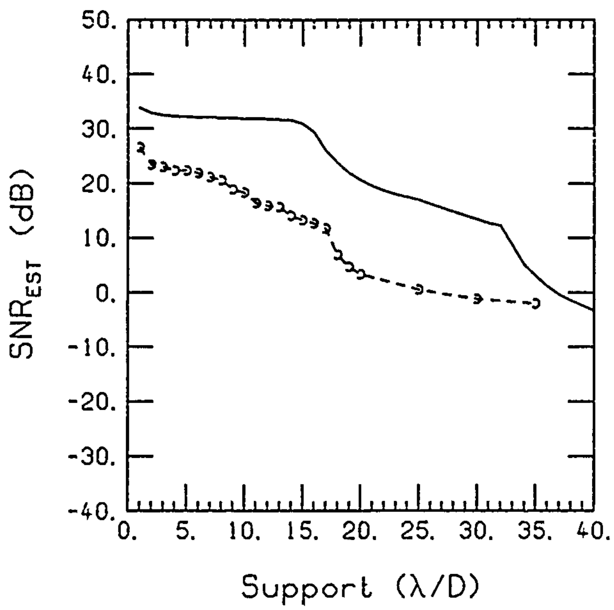


Figure 6.21.
Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The six-subaperture array of Fig. 6.4 was used with $SNR_{REF} = 60$ dB, $d/D = 0.005$, and $\kappa_0 = 0.76D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, $\mathbf{y}$, used here had a length of 256 λ/D.

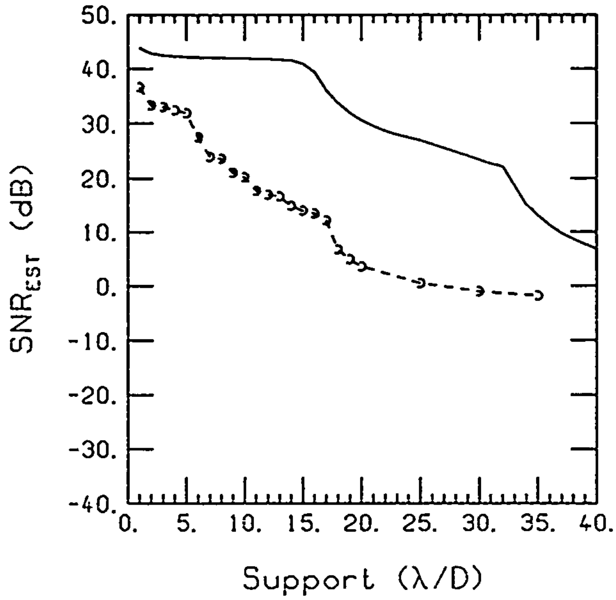


Figure 6.22.
Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The six-subaperture array of Fig. 6.4 was used with SNR_{REF} = 70 dB, $d/D = 0.005$, and $\kappa_0 = 0.76D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, **y**, used here had a length of 256 λ/D.

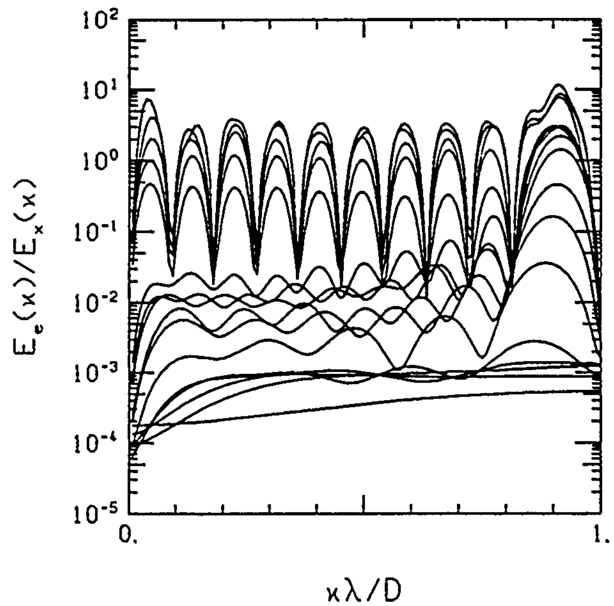


Figure 6.23.
Ratio of the energy spectrum of the error to the energy spectrum of the object versus spatial frequency κ using the CLEAN algorithm and the five-subaperture array of Fig. 6.3 with SNR_{REF} = 70 dB and $d/D = 0.005$. From bottom to top, the sixteen curves correspond to object supports of $1\lambda/D$, $2\lambda/D$, $\cdots$, $12\lambda/D$, $15\lambda/D$, $20\lambda/D$, $25\lambda/D$, and $30\lambda/D$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, **y**, used here had a length of 256 λ/D.

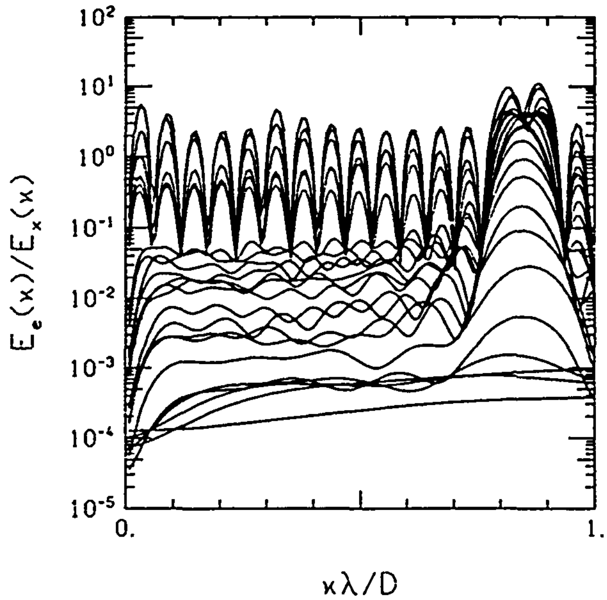
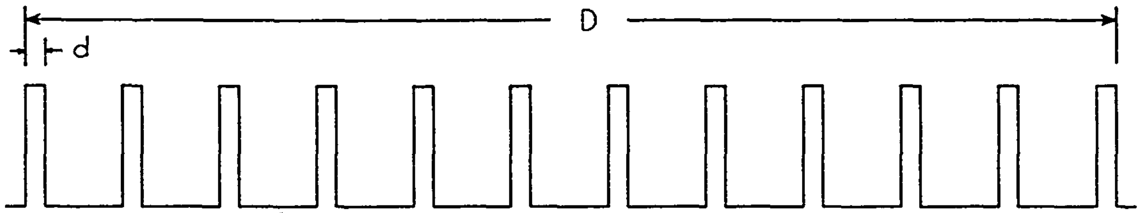


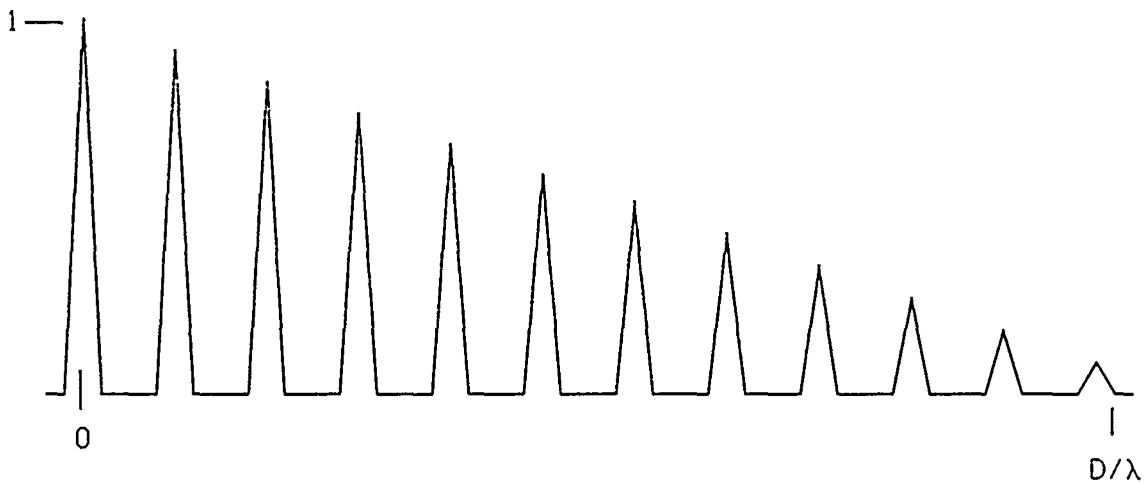
Figure 6.24.

Ratio of the energy spectrum of the error to the energy spectrum of the object versus spatial frequency κ using the CLEAN algorithm and the six-subaperture array of Fig. 6.4 with $SNR_{REF} = 70$ dB and $d/D = 0.005$. From bottom to top, the twenty three curves correspond to object supports of $1\lambda/D$, $2\lambda/D$, $\cdots$, $20\lambda/D$, $25\lambda/D$, $30\lambda/D$, and $35\lambda/D$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, $\mathbf{y}$, used here had a length of 256 λ/D.

with and without positivity, we have "allowed" the algorithm to use as much of th^e image line as was necessary in order to maximize performance. In other words, from a standpoint of performance, one can consider the dimension of the observation vector $\mathbf{y}$ to be infinite. In experiments with CLEAN, we found that the performance did not monotonically improve as the size of the image line was increased beyond the size of the object. In fact, performance was erratic beyond that point, changing by as much as ± 4 dB in the region to the left of the "cliff" in the SNR_{EST} curve. It appears that CLEAN can not effectively utilize object information contained in the outlying region of the image line. Accordingly, our results are approximately equivalent to what we would have gotten if we had formulated the CLEAN algorithm to only use image measurement data from the set of pixels at the center of the image plane and having an extent just matching the actual size of the target object. It is as though we had limited the size of the observation vector used by CLEAN to match the object support. This leads to the question as to how least-squares would perform compared to CLEAN if the size of the observation vector was limited to object support for both algorithms. The answer is contained in Fig. 6.33. There is no difference in performance. We conclude from this that the difference in performance, for the smaller object supports, seen in Fig.'s 6.14 to 6.22 and 6.27 to 6.32 is due to the different "effective" size of the observation vectors for the two algorithms. It is CLEAN's inability to make use of the multiplicity of copies of the image that the array produces that causes the difference. It is not clear that this inability would manifest itself in processing radio astronomy data.



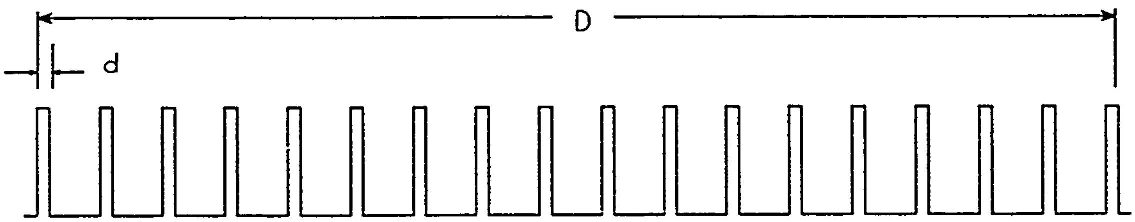
(a)



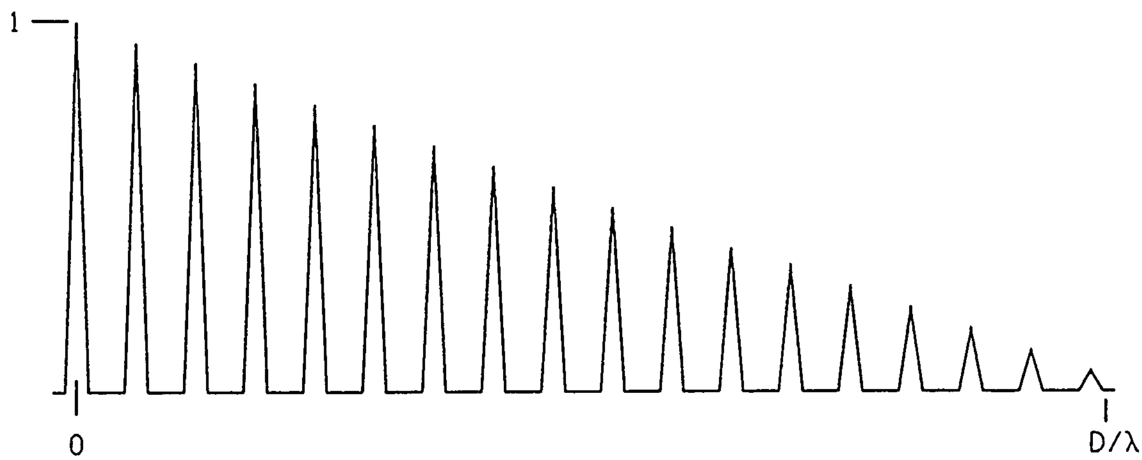
(b)

Figure 6.25.

Sparse array aperture function (a) and the corresponding MTF (b) for a redundant twelve- subaperture array.



(a)



(b)

Figure 6.26.

Sparse Array aperture function (a) and the corresponding MTF (b) for a redundant eighteen-subaperture array.

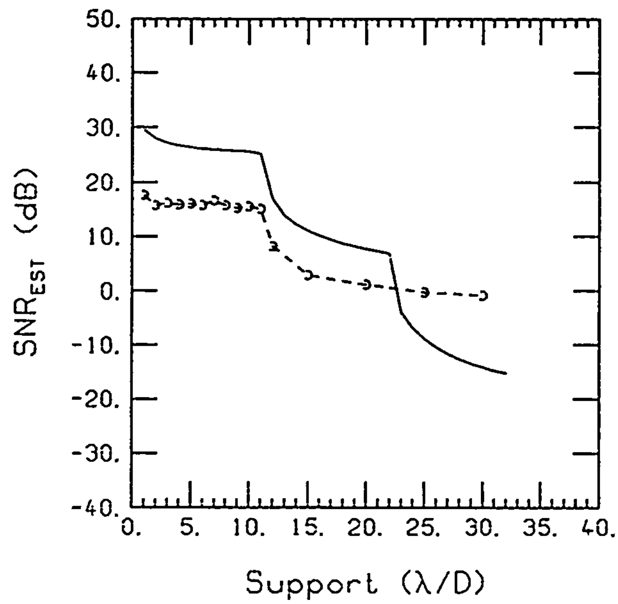


Figure 6.27.
Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The twelve-subaperture array of Fig. 6.25 was used with $\mathrm{SNR_{REF}} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 1.0D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, **y**, used here had a length of 256 λ/D for the least-squares results and a length equal to object support for the CLEAN results.

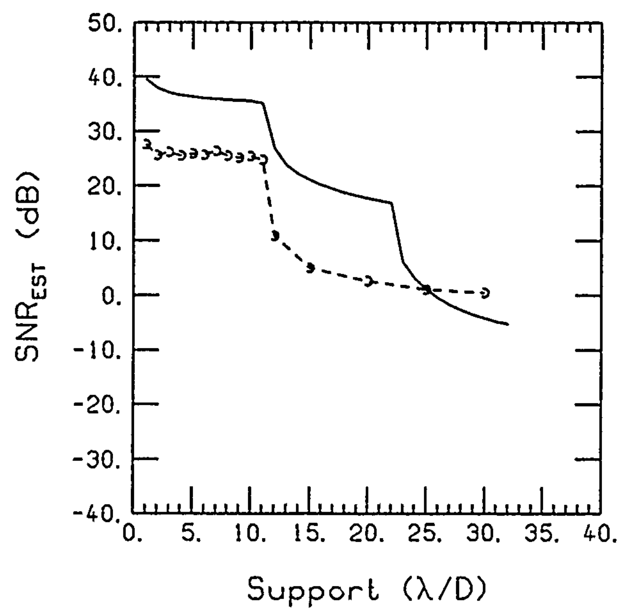


Figure 6.28.
Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The twelve-subaperture array of Fig. 6.25 was used with $\mathrm{SNR_{REF}} = 60$ dB, $d/D = 0.005$, and $\kappa_0 = 1.0D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, **y**, used here had a length of 256 λ/D for the least-squares results and a length equal to object support for the CLEAN results.

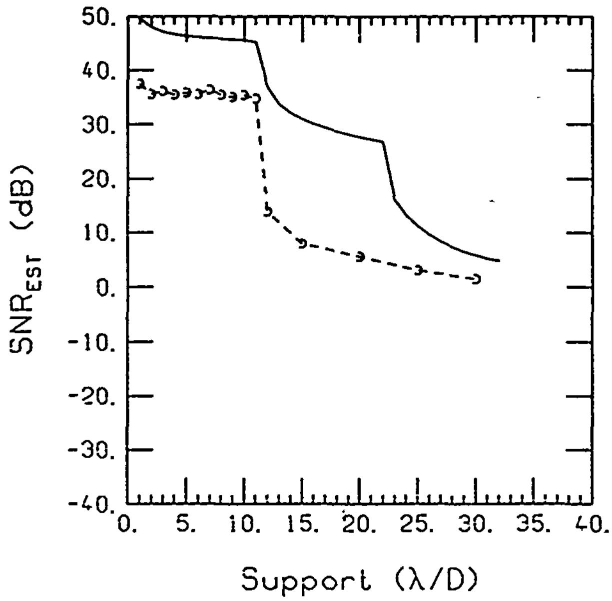


Figure 6.29.
Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The twelve-subaperture array of Fig. 6.25 was used with $\text{SNR}_{\text{REF}} = 70$ dB, $d/D = 0.005$, and $\kappa_0 = 1.0D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, **y**, used here had a length of 256 λ/D for the least-squares results and a length equal to object support for the CLEAN results.

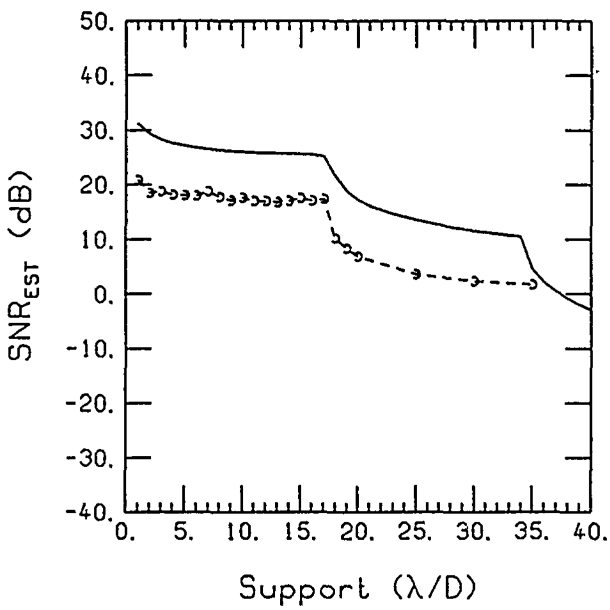


Figure 6.30.
Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The eighteen supaperture array of Fig. 6.26 was used with $\text{SNR}_{\text{REF}} = 50$ dB, $d/D = 0.005$, and $\kappa_0 = 1.0D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, **y**, used here had a length of 256 λ/D for the least-squares results and a length equal to object support for the CLEAN results.

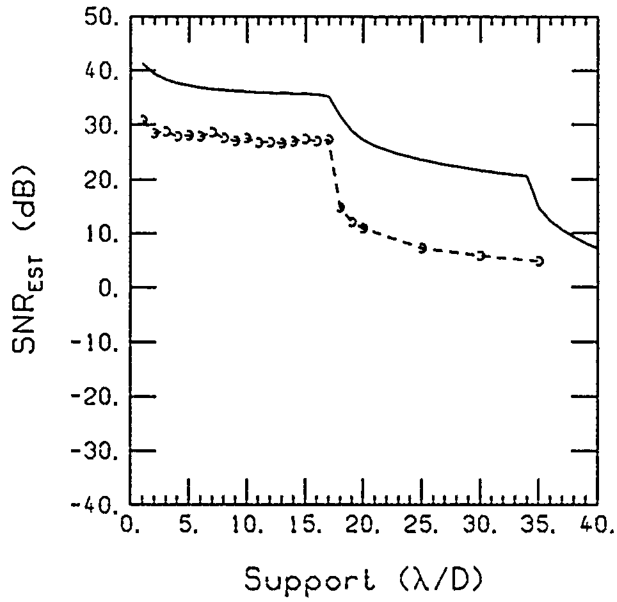


Figure 6.31.
Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The eighteen-subaperture array of Fig. 6.31 was used with $SNR_{REF} = 60$ dB, $d/D = 0.005$, and $\kappa_0 = 1.0D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, **y**, used here had a length of 256 λ/D for the least-squares results and a length equal to object support for the CLEAN results.

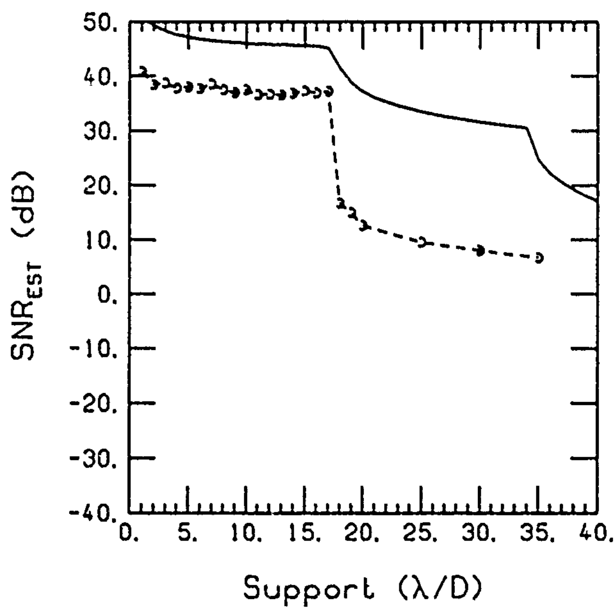


Figure 6.32.
Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The eighteen-subaperture array of Fig. 6.26 was used with $SNR_{REF} = 70$ dB, $d/D = 0.005$, and $\kappa_0 = 1.0D/\lambda$. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here. The image data array, **y**, used here had a length of 256 λ/D for the least-squares results and a length equal to object support for the CLEAN results.

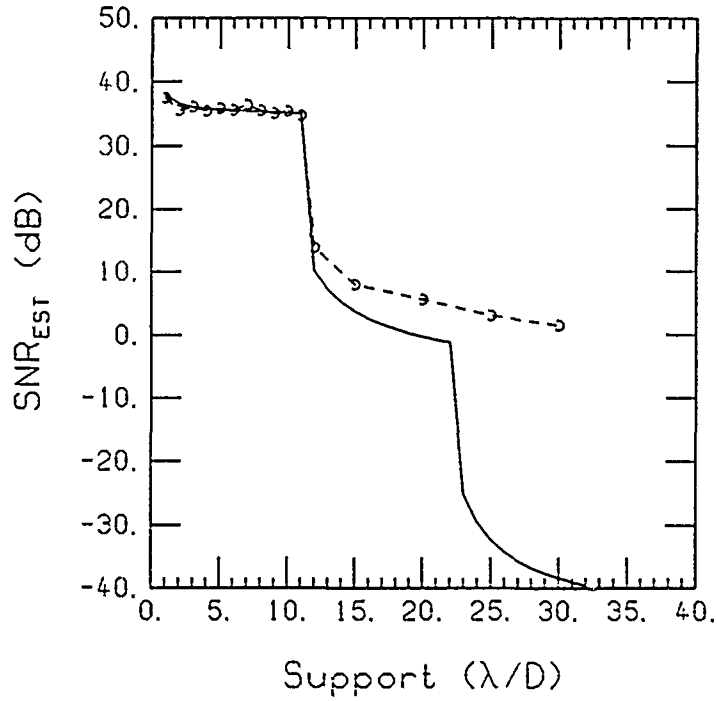


Figure 6.33.

Signal-to-noise ratio of the object estimate versus both assumed and actual object support using unweighted least-squares without a positivity constraint on the object estimate (solid line) and the CLEAN algorithm (circles). The twelve-subaperture array of Fig. 25 was used with $SNR_{REF} = 70$ dB, $d/D = 0.005$, and $\kappa_0 = 1.0D/\lambda$. In all cases, i.e., for both CLEAN and least-squares results, the length of the image line was the same as the length of the object support. There were $N = 40$ Monte Carlo runs used to generate the simulation data shown here.

REFERENCES

1. I. J. Cox and C. J. R. Sheppard, "Information Capacity and Resolution in an Optical System," J. Opt. Soc. Am. A 3, 1152–1158 (1987).
2. A. R. Thompson, J. M. Moran, and G. W. Swenson, Jr., *Interferometry and Synthesis in Radio Astronomy*, John Wiley and Sons, New York (1986).
3. R. W. Schafer, R. M. Mersereau, and M. A. Richards, "Constrained Iterative Restoration Algorithms," Proc. IEEE 69, 432–450 (1981).
4. R. Barakat, "Dilute Aperture Diffraction Imagery and Object Reconstruction," Opt. Eng. 29, 131–139 (1990).
5. W. Lukosz, "Optical Systems with Resolving Power Exceeding the Classical Limit," J. Opt. Soc. Am. 56, 1463–1472 (1966).
6. W. Lukosz, "Optical Systems with Resolving Power Exceeding the Classical Limit II," J. Opt. Soc. Am. 57, 932–941 (1967).
7. C. W. Barnes, "Object Restoration in a Diffraction-Limited Imaging System," J. Opt. Soc. Am. 56, 575–578 (1966).
8. B. R. Frieden, "Band-Unlimited Reconstruciton of Optical Objects and Spectra," J. Opt. Soc. Am. 57, 1013–1019 (1967).
9. J. L. Harris, "Diffraction and Resolving Power," J. Opt. Soc. Am. 54, 931–936 (1964).
10. C. K. Rushforth and R. W. Harris, "Restoration, Resolution, and Noise," J. Opt. Soc. Am. 58, 539–545 (1968).
11. H. A. Brown, "Effect of Truncation on Image Enhancement by Prolate Spheroidal Functions," J. Opt. Soc. Am. 59, 228–229 (1969).
12. G. T. Herman and D. Ro, "Image Recovery Using Iterative Data Refinement with Relaxation," Opt. Eng. 29, 513–523 (1990).
13. J. R. Fienup, "Reconstruciton of an Object From the Modulus of Its Fourier Transform," Opt. Lett. 3, 27–29 (1978).
14. J. R. Fienup, "Phase Retrieval Algorithms: A Comparison," Appl. Opt. 21, 2758–2769 (1982).
15. S. J. Reeves and R. M. Mersereau, "Optimal Estimation of the Regularization Parameter and Stabilizing Functional for Regularized Image Restoration," Opt. Eng. 29, 446–454 (1990).
16. R. Barakat, "Phase Retrieval in Two Dimensions," RADC-TR-88-251, 26–67 (1988).
17. F. T. S. Yu, *Optics and Information Theory*, John Wiley and Sons, New York, 141–168 (1976).
18. G. Toraldo di Francia, "Resolving Power and Information," J. Opt. Soc. Am. 45, 497 (1955).
19. H. S. Coleman and M. F. Coleman, "Theoretical Resolution Angles for Point and Line Test Objects in the Presence of a Luminous Background," J. Opt. Soc. Am. 37, 572 (1947).
20. V. Ranchi, *Optics, The Science of Vision*, New York University Press, New York, (1957).
21. V. Ranchi, "Resolving Power of Calculated and Detected Images," J. Opt. Soc. Am. 51, 458 (1961).
22. G. Toraldo di Francia, Nuovo Cimento Suppl. 3 IX, 426–438 (1952).
23. P. B. Fellgett and E. H. Linfoot, Proc. R. Soc. London Ser. A 247, 369–407 (1955).
24. A. Lannes, S. Roques, and M. J. Casanove, "Stabilized Reconstruction in Signal and Image Processing; Part I: Partial Deconvolution and Spectral Extrapolation with Limited Field," J. Mod. Opt. 34, 161–226 (1987).
25. A. Lannes, M. J. Casanove, and S. Roques, "Stabilized Reconstruction in Signal and Image Processing; Part II: Iterative Reconstruction With and Without Constraint. Interactive Implementation," J. Mod. Opt. 34, 321–370 (1987).
26. A. Lannes, S. Roques, and M. J. Casanove, "Resolution and Robustness in Image Processing: A New Regularization Principle," J. Opt. Soc. Am. 4, 189–199 (1987).
27. J. A. Roberts, Editor, *Indirect Imaging*, Proceedings of an International Symposium, Sydney Australia, 30 August to 2 September 1983, Cambridge University Press, London (1984).
28. D. T. Sherwood and D. L. Fried, "Object Reconstruction with Sparse Arrays of Optical Apertures. Part I, Linear Methods," (tOSC Report No. TR-1070, May 1990).

29. R. L. Fox, *Optimization Methods for Engineering Design*, Addison-Wesley, Reading Massachusetts, 1971, p. 196.

APPENDIX A for Chapter 2

Computer Listings

This appendix contains the Fortran source code of the computer programs used to generate the data for this chapter.

```
c
c
c         program 2dmtfmkr.f
c
c
          real*4 z1(128,128)
          d=1.0
          call eight(d,z1)
         stop
         end
c
c
c
               subroutine eight(d,z1)
               real*8 ha,a,b,c
               real*4 z(64,64),z1(128,128),dx,dy
               complex*8 berr(128,128)
               a=0
               b=2/3.*3.14159265
               c=(4/3.)*3.14159265
               do 150 jj=0,63
                tk=-(jj-32)/32.
                do 200 j=0,63
                 th=(j-32)/32.
                 call adder(a,a,d,tk,th,hh1)
                 call adder(b,a,d,tk,th,hh2)
                 call adder(a,b,d,tk,th,hh7)
                 call adder(a,c,d,tk,th,hh3)
                 call adder(b,c,d,tk,th,hh5)
                 call adder(c,a,d,tk,th,hh8)
                 call adder(c,b,d,tk,th,hh9)
                 if(d.eq.1)then
                  z(j+1,jj+1)=hh1
                 else
                  z(j+1,jj+1)=3*hh1+hh2+hh3+hh5+hh7+
     c            hh8+hh9
                 endif
200             continue
150            continue
               call solidplt('2dmtfd1.0',64,64,z)
               do 205 ii=1,128
                do 205 iii=1,128
205              berr(ii,iii)=0
               do 210 ii=1,64
                do 220 iii=1,64
                 berr(32+ii,32+iii)=z(ii,iii)
220             continue
210            continue
               dx=1
               dy=1
               call fft2d(berr,128,dx,dy)
               do 300 ii=1,128
                do 310 iii=1,128
                 z1(ii,iii)=real(berr(ii,iii))
310             continue
300            continue
               call solidplt('2dpsfd1.0',128,128,z1)
               return
               end
c
```

```
      function ha(d,de)
      real*8 ha
      ha=.5*(d**2)*(1.57079632-asin(de/d)-
     c(de/d)*sqrt(1-(de/d)**2))/.78539816
      return
      end
c
      subroutine adder(p1,p2,d,tk,th,hh)
      real*8 ha,p1,p2
      r1=(1-d)/2*(cos(p1)-cos(p2))+th
      r2=(1-d)/2*(sin(p1)-sin(p2))+tk
      de=sqrt(r1**2+r2**2)
      if(de.le.d)then
       hh=ha(d,de)
      else
       hh=0
      endif
      return
      end
```

```
c
c
c        program first.f
c
c
         parameter(nn=157,kk=68)
         real*8 gx(kk),gy(kk)
         integer idummy(1)
c
         n=157
         m=4
         m2=(m**2)*(m**2+1)/4
         l=31
         k=63
         d=.1
         call bufasg(idummy,128*n*m,ia)
         call bufasg(idummy,2*128*128,ib)
         call zero(idummy(ia),n,m,idummy(ib),d,gx,gy,m2)
         call bufrel(idummy(ib))
         call bufrel(idummy(ia))
         call plotfl('g14',m2,'d',gx,'d',gy)
        stop
        end
c
c
        subroutine zero(bw,n,m,z1,d,gx,gy,m2)
        real*8 bw(64,n,m),z1(128,128),gx(m2),gy(m2)
        call eight(d,z1)
        call one(bw,n,m,z1,d)
        call two(gx,gy,bw,n,m,m2)
        return
        end
c
c
         subroutine one(bw,n,m,z1,d)
         real*8 z1(128,128),bw(64,n,m)
         do 500 i=1,64
         do 501 ii=1,n
          do 502 iii=1,m
           bw(i,ii,iii)=0
502       continue
501      continue
500      continue
         do 510 i=1,64
          do 520 ii=1,127
           bw(i,ii,1)=z1(64+i,ii+1)
520       continue
          do 530 iii=2,m
           do 540 iv=1,127
            bw(i,iii+iv-1,iii)=bw(i,iii+iv-2,iii-1)
540        continue
530       continue
510      continue
         return
         end
c
c
c
         subroutine two(gx,gy,bw,n,m,m2)
         parameter(mm=4)
         real*8 gl(mm,mm),gb(mm,mm),gx(m2),gy(m2),bw(64,n,m)
         iw=1
         do 600 i=1,m
          do 605 iii=1,m
           do 605 iv=1,m
605         gb(iii,iv)=0
          j=1
          ia=i
```

```
      do 610 ii=1,64
       call four(bw,ii,abs(ia-ii)+1,gl,m,n)
       do 620 iii=1,m
        do 630 iv=1,m
         gb(iii,iv)=gb(iii,iv)+gl(iii,iv)
630     continue
620    continue
610   continue
      jg=m*(j-1)
      ig=m*(ia-1)
      if(ia.ne.j)then
       do 640 k1=1,m
        do 650 k2=1,m
         if(iw.gt.0)then
          gx(iw)=gb(k1,k2)
          iw=-iw
         else
          gy(abs(iw))=gb(k1,k2)
          iw=-iw+1
         endif
650     continue
640    continue
      else
       do 645 k1=1,m
        do 655 k2=k1,m
         if(iw.gt.0)then
          gx(iw)=gb(k1,k2)
          iw=-iw
         else
          gy(abs(iw))=gb(k1,k2)
          iw=-iw+1
         endif
655     continue
645    continue
      endif
      do 680 ix=2,m-i+1
       j=j+1
       jg=m*(j-1)
       ia=ia+1
       ig=m*(ia-1)
       call four(bw,j,ia,gl,m,n)
       if(j.ne.ia)then
        do 660 k3=1,m
         do 670 k4=1,m
          gb(k3,k4)=gb(k3,k4)+gl(k3,k4)
          if(iw.gt.0)then
           gx(iw)=gb(k3,k4)
           iw=-iw
          else
           gy(abs(iw))=gb(k3,k4)
           iw=-iw+1
          endif
670      continue
660     continue
       else
        do 665 k3=1,m
         do 675 k4=k3,m
          gb(k3,k4)=gb(k3,k4)+gl(k3,k4)
          if(iw.gt.0)then
           gx(iw)=gb(k3,k4)
           iw=-iw
          else
           gy(abs(iw))=gb(k3,k4)
           iw=-iw+1
          endif
675      continue
665     continue
       endif
```

```
 680     continue
 600    continue
        return
        end
c
c
c
c
        subroutine four(bw,j,i,gl,m,n)
        real*8 gl(m,m),bw(64,n,m)
        do 400 k1=1,m
         do 410 k2=1,m
          gl(k1,k2)=0
          do 420 k3=1,n
           gl(k1,k2)=gl(k1,k2)+bw(j,k3,k1)*bw(i,k3,k2)
 420      continue
 410     continue
 400    continue
        return
        end
c
c
c
c
c
c
             subroutine eight(d,z1)
             real*8 ha,a,b,c,z1(128,128)
             real z(64,64),dx,dy
             complex*8 berr(128,128)
             a=0
             b=2/3.*3.14159265
             c=(4/3.)*3.14159265
             do 150 jj=0,63
              tk=-(jj-32)/32.
              do 200 j=0,63
               th=(j-32)/32.
               call adder(a,a,d,tk,th,hh1)
               call adder(b,a,d,tk,th,hh2)
               call adder(a,b,d,tk,th,hh7)
               call adder(a,c,d,tk,th,hh3)
               call adder(b,c,d,tk,th,hh5)
               call adder(c,a,d,tk,th,hh8)
               call adder(c,b,d,tk,th,hh9)
               if(d.eq.1)then
                z(j+1,jj+1)=hh1
               else
                z(j+1,jj+1)=3*hh1+hh2+hh3+hh5+hh7+
     c          hh8+hh9
               endif
 200          continue
 150         continue
             call solidplt('solidfile',64,64,z)
             do 205 ii=1,128
              do 205 iii=1,128
 205           berr(ii,iii)=0
             do 210 ii=1,64
              do 220 iii=1,64
               berr(32+ii,32+iii)=z(ii,iii)
 220          continue
 210         continue
             dx=1
             dy=1
             call fft2d(berr,128,dx,dy)
             do 300 ii=1,128
              do 310 iii=1,128
               z1(ii,iii)=real(berr(ii,iii))
 310          continue
```

```
  300       continue
            return
            end
c
            function ha(d,de)
            real*8 ha
            ha=.5*(d**2)*(1.57079632-asin(de/d)-
     c      (de/d)*sqrt(1-(de/d)**2))/.78539816
            return
            end
c
            subroutine adder(p1,p2,d,tk,th,hh)
            real*8 ha,p1,p2
            r1=(1-d)/2*(cos(p1)-cos(p2))+th
            r2=(1-d)/2*(sin(p1)-sin(p2))+tk
            de=sqrt(r1**2+r2**2)
            if(de.le.d)then
             hh=ha(d,de)
            else
             hh=0
            endif
            return
            end
c
c
```

```
c
c
c        program matrix4.f
c
c
         parameter(nn=157,mm=8,kk=64)
         real*8 z1(128,128),g(kk,kk),bw(64,nn,mm)
         integer lwork(kk),mwork(kk)
c
         n=157
         m=8
         m2=m**2
         l=31
         k=63
         d=1
         call eight(d,z1)
         call one(bw,k,n,m,z1,d)
         call two(g,bw,n,m,m2)
c        open(1,file='out1')
         snr=10000
         do 120 i=1,m2
          g(i,i)=g(i,i)+1/snr
120      continue
         call dminvw(g,m2,ier,lwork,mwork)
         do 130 iii=1,m2
          do 140 iv=1,m2
           g(iii,iv)=g(iii,iv)/snr
140       continue
130      continue
c          write(1,*)g
c          close(1)
        call ten(g,m,m2)
        stop
        end
c
c
c
c
         subroutine one(bw,k,n,m,z1,d)
         real*8 z1(128,128),bw(64,n,m)
         do 500 i=1,64
         do 501 ii=1,n
          do 502 iii=1,m
           bw(i,ii,iii)=0
502       continue
501      continue
500      continue
         do 510 i=1,64
          do 520 ii=1,127
           bw(i,ii,1)=z1(64+i,ii+1)
520       continue
          do 530 iii=2,m
           do 540 iv=1,127
            bw(i,iii+iv-1,iii)=bw(i,iii+iv-2,iii-1)
540        continue
530       continue
510      continue
         return
         end
c
c
c
         subroutine two(g,bw,n,m,m2)
         parameter(mm=8)
         real*8 gl(mm,mm),gb(mm,mm),g(m2,m2),bw(64,n,m)
         do 600 i=1,m
          do 605 iii=1,m
           do 605 iv=1,m
```

```
 605      gb(iii,iv)=0
        j=1
        ia=i
        do 610 ii=1,64
         call four(bw,ii,abs(ia-ii)+1,gl,m,n)
         do 620 iii=1,m
          do 630 iv=1,m
           gb(iii,iv)=gb(iii,iv)+gl(iii,iv)
 630      continue
 620     continue
 610    continue
        jg=m*(j-1)
        ig=m*(ia-1)
        do 640 k1=1,m
         do 650 k2=1,m
          g(jg+k1,ig+k2)=gb(k1,k2)
          g(ig+k2,jg+k1)=gb(k1,k2)
 650     continue
 640    continue
        do 680 ix=2,m-i+1
         j=j+1
         jg=m*(j-1)
         ia=ia+1
         ig=m*(ia-1)
         call four(bw,j,ia,gl,m,n)
         do 660 k3=1,m
          do 670 k4=1,m
           gb(k3,k4)=gb(k3,k4)+gl(k3,k4)
           g(jg+k3,ig+k4)=gb(k3,k4)
           g(ig+k4,jg+k3)=gb(k3,k4)
 670      continue
 660     continue
 680    continue
 600    continue
        return
        end
c
c
c
c
        subroutine four(bw,j,i,gl,m,n)
        real*8 gl(m,m),bw(64,n,m)
        do 400 k1=1,m
         do 410 k2=1,m
          gl(k1,k2)=0
          do 420 k3=1,n
           gl(k1,k2)=gl(k1,k2)+bw(j,k3,k1)*bw(i,k3,k2)
 420      continue
 410     continue
 400    continue
        return
        end
c
c
c
c
c
c
             subroutine eight(d,z1)
             real*8 ha,a,b,c,z1(128,128)
             real z(64,64),dx,dy
             complex*8 berr(128,128)
             a=0
             b=2/3.*3.14159265
             c=(4/3.)*3.14159265
             do 150 jj=0,63
              tk=-(jj-32)/32.
              do 200 j=0,63
```

```
                  th=(j-32)/32.
                  call adder(a,a,d,tk,th,hh1)
                  call adder(b,a,d,tk,th,hh2)
                  call adder(a,b,d,tk,th,hh7)
                  call adder(a,c,d,tk,th,hh3)
                  call adder(b,c,d,tk,th,hh5)
                  call adder(c,a,d,tk,th,hh8)
                  call adder(c,b,d,tk,th,hh9)
                  if(d.eq.1)then
                   z(j+1,jj+1)=hh1
                  else
                   z(j+1,jj+1)=3*hh1+hh2+hh3+hh5+hh7+
     c             hh8+hh9
                  endif
200              continue
150             continue
c               call solidplt('solidfile',64,64,z)
                do 205 ii=1,128
                 do 205 iii=1,128
205               berr(ii,iii)=0
                do 210 ii=1,64
                 do 220 iii=1,64
                  berr(32+ii,32+iii)=z(ii,iii)
220              continue
210             continue
                dx=1
                dy=1
                call fft2d(berr,128,dx,dy)
                do 300 ii=1,128
                 do 310 iii=1,128
                  z1(ii,iii)=real(berr(ii,iii))
310              continue
300             continue
                return
                end
c
                function ha(d,de)
                real*8 ha
                ha=.5*(d**2)*(1.57079632-asin(de/d)-
     c          (de/d)*sqrt(1-(de/d)**2))/.78539816
                return
                end
c
                subroutine adder(p1,p2,d,tk,th,hh)
                real*8 ha,p1,p2
                r1=(1-d)/2*(cos(p1)-cos(p2))+th
                r2=(1-d)/2*(sin(p1)-sin(p2))+tk
                de=sqrt(r1**2+r2**2)
                if(de.le.d)then
                 hh=ha(d,de)
                else
                 hh=0
                endif
                return
                end
c
c
c                fft program
          subroutine ten(g,m,m3)
          parameter(mm=8)
          real*4 diag(15),bdiag(128),wr(128),wi(128),dd,br(mm,mm,127)
          real*4 bi(mm,mm,127),y(128),sur(127,127),sli(16129),x(16129)
          real*8 gl(mm,mm),brt(mm,mm),bit(mm,mm),g(m3,m3)
          real*4 smsur(63,63),x2(3969),sli2(3969)
c
          n=128
          nl=7
          m2=2*m-1
```

```
      call lookup(n,wr,wi)
      do 10 i=1,m
       do 20 ii=i,m
        do 30 iii=1,m
         do 40 iv=1,m
          gl(iii,iv)=g(m*i-m+iii,m*ii-m+iv)
40       continue
30      continue
        call addiag(gl,m2,diag,m)
        call pad(diag,m2,bdiag,n)
        do 50 ix=1,n
         y(ix)=0
50      continue
        dd=1
        call fft2(bdiag,y,n,nl,wr,wi,dd)
        do 60 iii=1,n-1
         br(i,ii,iii)=bdiag(iii+1)
         br(ii,i,iii)=bdiag(iii+1)
         bi(i,ii,iii)=y(iii+1)
         bi(ii,i,iii)=-y(iii+1)
60      continue
20     continue
10    continue
      do 70 i=1,n-1
       do 80 ii=1,m
        do 90 iii=1,m
         brt(ii,iii)=br(ii,iii,i)
         bit(ii,iii)=bi(ii,iii,i)
90      continue
80     continue
       call addiag(brt,m2,diag,m)
       call pad(diag,m2,bdiag,n)
       call addiag(bit,m2,diag,m)
       call pad(diag,m2,y,n)
       dd=1
       call fft2(bdiag,y,n,nl,wr,wi,dd)
       do 100 ii=1,n-1
        sur(i,ii)=bdiag(ii+1)/m3
        sli((n-1)*i-(n-1)+ii)=bdiag(ii+1)/m3
        x((n-1)*(i-1)+ii)=(n-1)*(i-1)+ii
100    continue
70    continue
      iii=0
      do 110 i=1,63
       do 120 ii=1,63
        iii=iii+1
        smsur(i,ii)=sur(32+i,32+ii)
        sli2(iii)=smsur(i,ii)
        x2(iii)=x(iii)
120    continue
110   continue
      call solidplt('s24',127,127,sur)
      call plotfl('long24',(n-1)**2,'f',x,'f',sli)
      call solidplt('hi',63,63,smsur)
      call plotfl('lo',3969,'f',x2,'f',sli2)
      stop
      end
c
c
c
      subroutine pad(diag,m2,bdiag,n)
      real*4 diag(m2),bdiag(n)
      do 404 i=1,n
       bdiag(i)=0
404   continue
      k=(n-m2+3)/2
      do 406 i=1,m2
       bdiag(k+i-1)=diag(i)
```

```
 406  continue
      return
      end
c
c
c
      subroutine addiag(a,m2,diag,m)
      real*4 diag(m2)
      real*8 a(m,m)
      do 601 i=1,m2
       diag(i)=0
 601  continue
      do 603 i=1,m
       do 605 ii=1,m
        iii=i-ii+m
        diag(iii)=diag(iii)+a(i,ii)
 605   continue
 603  continue
      return
      end
```

```
c
c
c        program matrix5
c
c
         parameter(nn=254,mm=128)
         real*8 bw(nn,mm),retil(128),f(128)
         real*8 g(mm,mm)
         real*4 dx,dy,d
c
         nwt=1
         open(nwt,file='out')
         n=254
         l=128
         k=(n-l)/2.
         nl=256
         nl2=nl/2
         do 90 im=0,5
          m=2**(im+2)
         do 100 i=1,5
          d=-.1*i+.6
          call one(bw,k,n,m,d)
          do 110 ii=2,4
           snr=10**ii
           call two(bw,n,m,g,snr)
           call six(g,m,nl,retil,f,snr)
c          call five(retil,f,n,m,d,nwt,snr,nl2)
           if(im.eq.0)call plotfl('sparse4',nl2,'d',f,'d',retil)
           if(im.eq.1)call plotfl('sparse8',nl2,'d',f,'d',retil)
           if(im.eq.2)call plotfl('sparse16',nl2,'d',f,'d',retil)
           if(im.eq.3)call plotfl('sparse32',nl2,'d',f,'d',retil)
           if(im.eq.4)call plotfl('sparse64',nl2,'d',f,'d',retil)
           if(im.eq.5)call plotfl('sparse128',nl2,'d',f,'d',retil)
110       continue
100      continue
90       continue
         close(nwt)
         stop
         end
c
c
c
c
         subroutine one(bw,k,n,m,d)
c           subroutine to calculate bw
         real*8 bw(n,m),hh
         do 500 i=1,n
          do 500 ii=1,m
500        bw(i,ii)=0
         j=-k
         do 510 i=1,1+2*k
          bw(i,1)=hh(j,d)
          j=j+1
510      continue
         do 520 i=1,m-1
          do 530 ii=0,2*k
           bw(1+ii+i,i+1)=bw(ii+i,i)
530       continue
520      continue
         return
         end
c
c
         function hh(j,d)
         real*8 hh
         if(j.eq.0)then
          hh=d**2
         else
```

```
        hh=(cos(.78539816*j*(1-d))*sin(.78539816*d*j)/
     c  (.78539816*j))**2
       endif
       return
       end
c
c
       subroutine two(bw,n,m,g,snr)
c       subroutine to calculate h=(gtg)*snr + i inv
       parameter(mm=128)
       real*8 bw(n,m),g(m,m)
       integer lwork(mm),mwork(mm),ier
       do 600 i=1,m
        do 610 ii=1,m
         g(i,ii)=0
         do 620 iii=1,n
          g(i,ii)=g(i,ii)+bw(iii,i)*bw(iii,ii)*snr
620      continue
        if(i.eq.ii)g(i,ii)=g(i,ii)+1
610     continue
600    continue
       call dminvw(g,m,ier,lwork,mwork)
       return
       end
c
c
c
       subroutine five(retil,f,n,m,d,nwt,snr,nl2)
       real*8 retil(nl2),f(nl2)
       write(nwt,890)n,m,d,snr
890    format(' n = ',i4,' m = ',i4,' d = ',f10.5,' snr = ',
     c f12.5)
       do 900 i=1,nl2/2
        write(nwt,895)f(i),retil(i)
895     format(2f12.5)
900    continue
       return
       end
c
c
       subroutine six(g,m,nl,retil,f,snr)
       complex*8 berr(256,256)
       real*8 g(m,m),retil(128),f(128)
       do 200 i=1,nl
        do 200 ii=1,nl
200      berr(i,ii)=0
       n1=(nl-m)/2.
       do 210 i=1,m
        do 220 ii=1,m
         berr(n1+i,n1+ii)=g(i,ii)
220     continue
210    continue
       dx=1
       dy=1
       call fft2d(berr,nl,dx,dy)
       nl2=nl/2
       nl4=nl/4
       do 230 i=1,nl2
        retil(i)=real(berr(nl2+i,nl2-i+2))/m
        x=i
        f(i)=x/nl4
230    continue
       return
       end
c
c
```

```
c
c
c     program matrix6
c
      parameter(nn=254,mm=128)
      real*8 bw(nn,mm),h(mm,nn),retilu(128),f(128),x(mm),no(nn)
      real*8 g(mm,mm),erru(mm,mm),retil2(128),err2(mm,mm)
      real*8 retilc2(128),errc2(mm,mm)
      real*8 y(nn),gg(mm,mm)
      real*4 d,eps
      integer*4 random
      character dfile*16
c
      nwt=2
      open(nwt,file='out')
      n=254
      l=128
      k=(n-1)/2.
      print *, 'nt?'
      read *, nt
      print *, 'ni?'
      read *, ni
      print *, 'eps?'
      read *, eps
      print *, 'mb?'
      read *, mb
      print *, 'm?'
      read *, m
      print *, 'snr?'
      read *, snr
      print *, 'd/D?'
      read *, d
      print *, 'Destination File?'
      read *, dfile
      nl=256
      nl2=nl/2
      call srandom(29)
      call one(bw,k,n,m,d)
      call two(bw,h,n,m,g,gg)
      call two5(h,n,m,err2)
      call six(nwt,err2,m,nl,retil2,f,snr,mb)
      call five(retil2,f2,n,m,d,nwt,snr,nl2)
      call plotfl(dfile,nl2,'d',f,'d',retil2)
      do 103 j=1,m
       do 101 jj=1,m
        erru(j,jj)=0
        errc2(j,jj)=0
101    continue
103   continue
      do 105 iv=1,nt
c       print *,iv
       if (float(iv)/10.0-int(float(iv)/10.0) .lt. 1e-10) then
        print *, iv
       endif
       call three(x,no,m,n,snr,bw,y,mb)
       call four(x,h,no,erru,n,m,nt,errc2,y,bw,gg,eps,ni)
105   continue
      call six(nwt,erru,m,nl,retilu,f,snr,mb)
      call five(retilu,f,n,m,d,nwt,snr,nl2)
      call six(nwt,errc2,m,nl,retilc2,f,snr,mb)
      call five(retilc2,f,n,m,d,nwt,snr,nl2)
      call plotfl(dfile,nl2,'d',f,'d',retilu)
      call plotfl(dfile,nl2,'d',f,'d',retilc2)
      close(nwt)
      stop
      end
c
c
```

```
c
c
      subroutine one(bw,k,n,m,d)
      real*8 bw(n,m),hh
      do 500 i=1,n
       do 500 ii=1,m
500     bw(i,ii)=0
      j=-k
      do 510 i=1,1+2*k
       bw(i,1)=hh(j,d)
       j=j+1
510   continue
      do 520 i=1,m-1
       do 530 ii=0,2*k
        bw(1+ii+i,i+1)=bw(ii+i,i)
530    continue
520   continue
      return
      end
c
c
      function hh(j,d)
      real*8 hh
      if(j.eq.0)then
       hh=d**2
      else
       hh=(cos(.78539816*j*(1-d))*sin(.78539816*d*j)/
     c      (.78539816*j))**2
      endif
      return
      end
c
c
      subroutine two(bw,h,n,m,g,gg)
      parameter(mm=128)
      real*8 bw(n,m),g(m,m),h(m,n),gg(m,m)
      integer lwork(mm),mwork(mm),ier
      do 600 i=1,m
       do 610 ii=1,m
        g(i,ii)=0
        do 620 iii=1,n
         g(i,ii)=g(i,ii)+bw(iii,i)*bw(iii,ii)
         gg(i,ii)=g(i,ii)
620     continue
610    continue
600   continue
      call dminvw(g,m,ier,lwork,mwork)
      do 630 i=1,m
       do 640 ii=1,n
        h(i,ii)=0
        do 650 iii=1,m
         h(i,ii)=h(i,ii)+g(i,iii)*bw(ii,iii)
650     continue
640    continue
630   continue
      return
      end
c
c
      subroutine two5(h,n,m,err2)
      real*8 h(m,n),err2(m,m)
      do 400 i=1,m
       do 410 ii=1,m
        err2(i,ii)=0
        do 420 iii=1,n
         err2(i,ii)=err2(i,ii)+h(i,iii)*h(ii,iii)
420     continue
410    continue
```

```
 400     continue
         return
         end
 c
         subroutine three(x,no,m,n,snr,bw,y,mb)
         real*8 u1,u2,u3,s
         real*8 x(m),no(n),bw(n,m),y(n)
         integer*4 MAXINTV,random
         parameter(MAXINTV=2147483647)
         n2=n/2
         do 500 i=1,n2
 3        u1=real(random())/MAXINTV
          if(u1.gt.1.or.u1.eq.0)goto 3
 2        u2=real(random())/MAXINTV
          if(u2.gt.1.or.u2.eq.0)goto 2
          no(2*i-1)=sqrt(-2*log(u1))*cos(6.2831853*u2)
          no(2*i)=sqrt(-2*log(u1))*sin(6.2831853*u2)
 500     continue
         s=snr/2
         do 505 i=1,m
 4        u3=real(random())/MAXINTV
          if(u3.gt.1.or.u3.eq.0)goto 4
          x(i)=sqrt(-2*log(u3))*sqrt(s)
 505     continue
         do 507 i=mb+1,m
          x(i)=0
 507     continue
         do 510 i=1,n
          y(i)=0
          do 520 ii=1,m
           y(i)=y(i)+bw(i,ii)*x(ii)
 520      continue
           y(i)=y(i)+no(i)
 510     continue
         return
         end
 c
 c
         subroutine four(x,h,no,erru,n,m,nt,errc2,y,bw,gg,eps,ni)
         parameter(mm=128,nn=254)
         real*8 x(m),no(n),erru(m,m),h(m,n),eu(mm),sum
         real*8 ec2(mm),errc2(m,m),gg(m,m),gg2(mm,mm),v2(nn)
         real*8 tmp(mm),y(n),bw(n,m),z(mm),xg(mm),tem2(mm)
         real*4 eps
         do 700 i=1,m
          tmp(i)=0
          z(i)=0
          do 710 ii=1,n
           tmp(i)=tmp(i)+h(i,ii)*y(ii)
           z(i)=z(i)+bw(ii,i)*y(ii)*eps
 710      continue
 700     continue
         do 712 i=1,m
          do 713 ii=1,m
           if(i.eq.ii)then
            gg2(i,ii)=1-eps*gg(i,ii)
           else
            gg2(i,ii)=-eps*gg(i,ii)
           endif
 713      continue
 712     continue
         do 715 i=1,m
          eu(i)=tmp(i)-x(i)
 c        if(tmp(i).gt.0.and.x(i).gt.0)then
 c         xg(i)=tmp(i)
 c        else
 c         xg(i)=0
 c        endif
```

```
        xg(i)=x(i)
        ec2(i)=xg(i)-x(i)
715    continue
       jk=0
       do 770 iv=1,ni
        jk=jk+1
        do 740 i=1,m
         tem=0
         do 750 ii=1,m
          tem=tem+gg2(i,ii)*xg(ii)
750      continue
         tem2(i)=tem+z(i)
         if(tem2(i).lt.0)tem2(i)=0
740     continue
        do 760 i=1,m
         xg(i)=tem2(i)
         ec2(i)=xg(i)-x(i)
760     continue
c       if(jk.eq.100)then
c        jk=0
c        sum=0
c         do 745 i=1,n
c          v2(i)=0
c          do 746 ii=1,m
c           v2(i)=v2(i)+bw(i,ii)*xg(ii)
c46        continue
c          v2(i)=v2(i)-y(i)
c          sum=sum+v2(i)**2
c45       continue
c        do 775 i=1,m
c         print *,xg(i)
c75      continue
c        print *,iv,sum
c        print *
c       endif
770    continue
c       call plotfl('test',m,'d',xg,'d',xg)
       do 720 i=1,m
        do 730 ii=1,m
         erru(i,ii)=erru(i,ii)+eu(i)*eu(ii)/nt
         errc2(i,ii)=errc2(i,ii)+ec2(i)*ec2(ii)/nt
730     continue
720    continue
       return
       end
c
c
c
c
c
       subroutine five(retil,f,n,m,d,nwt,snr,nl2)
       real*8 retil(nl2),f(nl2)
       write(nwt,890)n,m,d,snr
890    format(' n = ',i4,' m = ',i4,' d = ',f10.5,' snr = ',
     c f16.4)
       do 900 i=1,nl2/2
        write(nwt,895)f(i),retil(i)
895     format(2f12.5)
900    continue
       return
       end
c
c
       subroutine six(nwt,err,m,nl,retil,f,snr,mb)
       complex*8 berr(256,256)
       real*8 err(m,m),retil(128),f(128)
       real*4 dx,dy
       parameter (pi=3.14159265)
```

```
            sum=0
            do 190 i=1,m
             sum=sum+err(i,i)
  190       continue
            write(nwt,*)'trace = ',sum
            print *, sum
            do 200 i=1,nl
             do 200 ii=1,nl
  200         berr(i,ii)=0
            n1=(nl-m)/2.
            do 210 i=1,m
             do 220 ii=1,m
              berr(n1+i,n1+ii)=err(i,ii)
  220        continue
  210       continue
            dx=1
            dy=1
            call fft2d(berr,nl,dx,dy)
            nl2=nl/2
            nl4=nl/4
            do 230 i=1,nl2
             f(i)=real(i)/nl4
             retil(i)=real(berr(nl2+i,nl2-i+2))/(snr*mb)
  230       continue
            return
            end
c
c
```

```
c
c
c      program matrix7
c
       parameter(nn=254,mm=128)
       real*8 bw(nn,mm),h(mm,nn),retilu(128),f(128),x(mm),no(nn)
       real*8 g(mm,mm),erru(mm,mm),retil2(128),err2(mm,mm)
       real*8 retilc2(128),errc2(mm,mm)
       real*8 y(nn),gg(mm,mm)
       real*4 d,eps
       integer*4 random
       character dfile*16
c
       n=254
       l=128
       k=(n-1)/2.
       print *, 'snr?'
       read *, snr
       print *, 'd/D?'
       read *, d
       print *, 'Destination File?'
       read *, dfile
       nl=256
       nl2=nl/2
       do 5 i=1,6
        m=2**(i+1)
        mb=m
        do 10 j=1,m
         do 11 jj=1,m
          erru(j,jj)=0
          errc2(j,jj)=0
11       continue
10      continue
        call one(bw,k,n,m,d)
        call two(bw,h,n,m,g,gg)
        call two5(h,n,m,err2)
        call six(nwt,err2,m,nl,retil2,f,snr,mb)
        call plotfl(dfile,nl2,'d',f,'d',retil2)
5      continue
       stop
       end
c
c
c
c
       subroutine one(bw,k,n,m,d)
       real*8 bw(n,m),hh
       do 500 i=1,n
        do 500 ii=1,m
500      bw(i,ii)=0
       j=-k
       do 510 i=1,1+2*k
        bw(i,1)=hh(j,d)
        j=j+1
510    continue
       do 520 i=1,m-1
        do 530 ii=0,2*k
         bw(1+ii+i,i+1)=bw(ii+i,i)
530     continue
520    continue
       return
       end
c
c
       function hh(j,d)
       real*8 hh
       if(j.eq.0)then
        hh=d**2
```

```
              else
               hh=(cos(.78539816*j*(1-d))*sin(.78539816*d*j)/
     c          (.78539816*j))**2
              endif
              return
              end
c
c
              subroutine two(bw,h,n,m,g,gg)
              parameter(mm=128)
              real*8 bw(n,m),g(m,m),h(m,n),gg(m,m)
              integer lwork(mm),mwork(mm),ier
              do 600 i=1,m
               do 610 ii=1,m
                g(i,ii)=0
                do 620 iii=1,n
                 g(i,ii)=g(i,ii)+bw(iii,i)*bw(iii,ii)
                 gg(i,ii)=g(i,ii)
620             continue
610            continue
600           continue
              call dminvw(g,m,ier,lwork,mwork)
              do 630 i=1,m
               do 640 ii=1,n
                h(i,ii)=0
                do 650 iii=1,m
                 h(i,ii)=h(i,ii)+g(i,iii)*bw(ii,iii)
650             continue
640            continue
630           continue
              return
              end
c
c
              subroutine two5(h,n,m,err2)
              real*8 h(m,n),err2(m,m)
              do 400 i=1,m
               do 410 ii=1,m
                err2(i,ii)=0
                do 420 iii=1,n
                 err2(i,ii)=err2(i,ii)+h(i,iii)*h(ii,iii)
420             continue
410            continue
400           continue
              return
              end
c
              subroutine six(nwt,err,m,nl,retil,f,snr,mb)
              complex*8 berr(256,256)
              real*8 err(m,m),retil(128),f(128)
              real*4 dx,dy
              parameter (pi=3.14159265)
              sum=0
              do 190 i=1,m
               sum=sum+err(i,i)
190           continue
              print *, sum
              do 200 i=1,nl
               do 200 ii=1,nl
200             berr(i,ii)=0
              n1=(nl-m)/2.
              do 210 i=1,m
               do 220 ii=1,m
                berr(n1+i,n1+ii)=err(i,ii)
220            continue
210           continue
              dx=1
              dy=1
```

```
           call fft2d(berr,nl,dx,dy)
           nl2=nl/2
           nl4=nl/4
           do 230 i=1,nl2
            f(i)=real(i)/nl4
            retil(i)=real(berr(nl2+i,nl2-i+2))/(snr*mb)
  230      continue
           return
           end
c
c
```

```fortran
c
c
c       program   mtfmkr.f
c
c
        real*4 xi,xj,xn,f,d,mtf,fr(200),o(200)
        integer n
        n = 200
        xn = n
        do 10 j=1, 5
         xj = j
         d = xj/10
         do 20 i=0, n-1
           xi = i
           f = xi/xn
           fr(i+1) = mtf(d,f) + 1e-30
           o(i+1) = f
 20      continue
         call plotfl('mtfdta',200,'f',o,'f',fr)
 10     continue
        end
c-----------------------------------------------------------------
         FUNCTION mtf(d,f)
         real*4 d,f,dinv,mtf,f1,f2
         dinv = 1/d
         if (f .lt. d) then
           f1 = 1.0 - dinv*f
         else
           f1 = 0
         endif
         if ((f .gt. 1-2*d) .and. (f .le. 1-d)) then
           f2 = 0.5*dinv*(f-1) + 1
         elseif ((f .gt. 1-d) .and. (f .le. 1)) then
           f2 = 0.5*dinv*(1-f)
         else
           f2 = 0
         endif
         mtf = f1 + f2
         mtf = 2.0*d*mtf
         end
```

```
c
c
c         program second.f
c
c
          parameter(kk=784)
          real*8 g(kk,kk)
          integer lwork(kk),mwork(kk),idummy(1)
c
          m=28
          m2=m**2
          m3=(m2*(m2+1))/4
          call bufasg(idummy,2*m3,ia)
          call bufasg(idummy,2*m3,ib)
          call three(g,m,m2,m3,idummy(ia),idummy(ib))
          call bufrel(idummy(ib))
          call bufrel(idummy(ia))
          snr=100
          do 120 i=1,m2
           g(i,i)=g(i,i)+1/snr
  120     continue
          call dminvv(g,m2,ier,lwork,mwork)
          do 130 iii=1,m2
           do 140 iv=1,m2
            g(iii,iv)=g(iii,iv)/snr
  140      continue
  130     continue
         call ten(g,m,m2)
         stop
         end
c
         subroutine three(g,m,m2,m3,gx,gy)
         real*8 g(m2,m2),gx(m3),gy(m3)
         call readfl('g328',n,'d',gx,'d',gy,1,1,m3)
         iw=1
         do 600 i=1,m
          j=1
          ia=i
          jg=m*(j-1)
          ig=m*(ia-1)
          do 680 ix=1,m-i+1
          if(ia.ne.j)then
           do 640 k1=1,m
            do 650 k2 = 1,m
             if(iw.gt.0)then
              g(jg+k1,ig+k2)=gx(iw)
              g(ig+k2,jg+k1)=gx(iw)
              iw=-iw
             else
              g(jg+k1,ig+k2)=gy(-iw)
              g(ig+k2,jg+k1)=gy(-iw)
              iw=-iw+1
             endif
  650       continue
  640      continue
          else
           do 645 k1=1,m
            do 655 k2=k1,m
             if(iw.gt.0)then
              g(jg+k1,ig+k2)=gx(iw)
              g(ig+k2,ig+k1)=gx(iw)
              iw=-iw
             else
              g(jg+k1,ig+k2)=gy(-iw)
              g(ig+k2,jg+k1)=gy(-iw)
              iw=-iw+1
             endif
  655       continue
```

```
645         continue
           endif
           j=j+1
           jg=m*(j-1)
           ia=ia+1
           ig=m*(ia-1)
680       continue
600       continue
          return
          end
c
c
          subroutine ten(g,m,m3)
          parameter(mm=28)
          real*4 diag(55),bdiag(128),wr(128),wi(128),dd,br(mm,mm,127)
          real*4 bi(mm,mm,127),y(128),sur(127,127),sli(16129),x(16129)
          real*8 gl(mm,mm),brt(mm,mm),bit(mm,mm),g(m3,m3)
          real*4 smsur(63,63)
c         real*4 x2(3969),sli2(3969)
c
          n=128
          nl=7
          m2=2*m-1
          call lookup(n,wr,wi)
          do 10 i=1,m
           do 20 ii=1,m
            do 30 iii=1,m
             do 40 iv=1,m
              gl(iii,iv)=g(m*i-m+iii,m*ii-m+iv)
40           continue
30          continue
            call addiag(gl,m2,diag,m)
            call pad(diag,m2,bdiag,n)
            do 50 ix=1,n
             y(ix)=0
50          continue
            dd=1
            call fft2(bdiag,y,n,nl,wr,wi,dd)
            do 60 iii=1,n-1
             br(i,ii,iii)=bdiag(iii+1)
c            br(ii,i,iii)=bdiag(iii+1)
             bi(i,ii,iii)=y(iii+1)
c            bi(ii,i,iii)=-y(iii+1)
60          continue
20         continue
10        continue
          do 70 i=1,n-1
           do 80 ii=1,m
            do 90 iii=1,m
             brt(ii,iii)=br(ii,iii,i)
             bit(ii,iii)=bi(ii,iii,i)
90          continue
80         continue
           call addiag(brt,m2,diag,m)
           call pad(diag,m2,bdiag,n)
           call addiag(bit,m2,diag,m)
           call pad(diag,m2,y,n)
           dd=1
           call fft2(bdiag,y,n,nl,wr,wi,dd)
           do 100 ii=1,n-1
            sur(i,ii)=bdiag(ii+1)/m3
            sli((n-1)*i-(n-1)+ii)=bdiag(ii+1)/m3
            x((n-1)*(i-1)+ii)=(n-1)*(i-1)+ii
100        continue
70        continue
          iii=0
          do 110 i=1,63
           do 120 ii=1,63
```

```
c          iii=iii+1
           smsur(i,ii)=sur(32+i,32+ii)
c          sli2(iii)=smsur(i,ii)
c          x2(iii)=x(iii)
  120     continue
  110    continue
c        call solidplt('sb28',127,127,sur)
         call plotfl('tlongb28a',(n-1)**2,'f',x,'f',sli)
         call solidplt('ttb28a',63,63,smsur)
c        call plotfl('slongb28',3969,'f',x2,'f',sli2)
         stop
         end
c
c
c
         subroutine pad(diag,m2,bdiag,n)
         real*4 diag(m2),bdiag(n)
         do 404 i=1,n
          bdiag(i)=0
  404    continue
         k=(n-m2+3)/2
         do 406 i=1,m2
          bdiag(k+i-1)=diag(i)
  406    continue
         return
         end
c
c
c
         subroutine addiag(a,m2,diag,m)
         real*4 diag(m2)
         real*8 a(m,m)
         do 601 i=1,m2
          diag(i)=0
  601    continue
         do 603 i=1,m
          do 605 ii=1,m
           iii=i-ii+m
           diag(iii)=diag(iii)+a(i,ii)
  605     continue
  603    continue
         return
         end
```

```
c
c
c        program slice.f
c
c
c        this program reads in the results of matrix4
c        in plotfl form and reassembles spectral matrix
c        and then takes out the center to plot
         real*4 x(16129),surf(127,127),smsur(63,63)
         real*4 x2(3969),sli2(3969),sli(16129)
         character*10 file
         print *,'What is the name of the file?'
         read *,file
         call readfl(file,n,'f',x,'f',sli,1,1,16129)
         i=0
         do 10 ii=1,127
          do 20 iii=1,127
           i=i+1
           surf(ii,iii)=sli(i)
20        continue
10       continue
c         i=0
c         do 30 ii=1,63
c          do 40 iii=1,63
c           i=i+1
c           smsurf(ii,iii)=surf(32+ii,32+iii)
c           sli2(i)=smsurf(ii,iii)
c           x2(i)=x(i)
c0         continue
c0        continue
          call solidplt('solidfile',127,127,surf)
c         call plotfl('smlong2',3969,'f',x2,'f',sli2)
          stop
          end
```

APPENDIX B for Chapter 3

Computer Listings

This appendix contains the Fortran source code of the computer programs used to generate the data for this chapter.

```
c
c
c          program eigenmkr.f
c
c
           parameter(mm=128)
           real*8 bw(1150,mm),f(128)
           real*8 g(mm,mm),retil1(128),retil2(128),g1(mm,mm),g2(mm,mm)
           real*8 gg(mm,mm),evec(mm,mm),eval(mm),vec(mm,mm),val(mm),d
           real*4 eps
           integer*4 i,k,m,mb,n,l,ier,window
           character dfile*16
c
           l=128
           print *, 'k?'
           read *, k
           print *, 'snr?'
           read *, snr
           print *, 'd/D?'
           read *, d
           print *, 'Window?'
           read *, window
c          print *, 'Destination File?'
c          read *, dfile
           n=2*k+128
           do 10 i=1,6
             m=2**(i+1)
             mb=m
             call bwmkr(bw,k,n,m,d,window)
             call ggmkr(bw,n,m,g,gg,eps,ier)
c            call eigenmkr(m,gg,evec,eval,vec,val,g1,g2)
             call eigenmkr(m,n,bw,evec,eval,vec,val,g1,g2)
             call spectramkr(g1,m,retil1,f,snr,mb)
             call spectramkr(g2,m,retil2,f,snr,mb)
             do 20 ii=1,128
c              retil2(ii)=retil2(ii)+retil1(ii)
 20          continue
c            call plotfl(dfile,128,'d',f,'d',retil2)
             call plotfl('testw3k255d1.1',128,'d',f,'d',retil1)
             call plotfl('testw3k255d1.2',128,'d',f,'d',retil2)
 10        continue
           stop
           end
c
c
c
           subroutine bwmkr(bw,k,n,m,d,window)
           real*8 bw(n,m),hh,d
           integer n,m,k,window
           do 500 i=1,n
            do 500 ii=1,m
 500         bw(i,ii)=0
           j=-k
           do 510 i=1,1+2*k
            bw(i,1)=hh(j,d,k,window)
            j=j+1
 510       continue
           do 520 i=1,m-1
            do 530 ii=0,2*k
             bw(1+ii+i,i+1)=bw(ii+i,i)
```

```
530     continue
520    continue
       return
       end
c
c
c
       subroutine ggmkr(bw,n,m,g,gg,eps,ier)
       parameter(mm=128)
       real*8 bw(n,m),g(m,m),gg(m,m)
       real*8 r(8256)
       real*4 eps
       integer ier
       do 600 i=1,m
        do 610 ii=1,m
         g(i,ii)=0
         do 620 iii=1,n
          g(i,ii)=g(i,ii)+bw(iii,i)*bw(iii,ii)
          gg(i,ii)=g(i,ii)
620      continue
610     continue
600    continue
c      call squartri(r,m,g)
c      call dsinv(r,m,eps,ier)
c      call trisquar(g,m,r)
       end
c
c
c
       function hh(j,d,k,window)
       real*8 hh,d,k1,pi,pi2,j1
       integer k,j,m,window
       m=window
       pi=3.141592654
       pi2=2*pi
       j1=j
       k1=k
       if(j.eq.0)then
        hh=d*d
       else
        hh=cos(.78539816*dble(j)*(1-d))*sin(.78539816*d*dble(j))/
     c  (.78539816*dble(j))
        hh=hh*hh
       endif
       if (m.eq.1) then
         hh=hh
        elseif (m.eq.2) then
         hh=hh*(1-abs(j1)/k1)
        elseif (m.eq.3) then
         hh=hh*(.5+.5*cos(pi*j1/k1))
        elseif (m.eq.4) then
         hh=hh*(0.54+0.46*cos(pi*j1/k1))
        else
         hh=hh*(0.42+0.5*cos(pi*j1/k1)+0.08*cos(pi2*j1/k1))
       endif
10     return
       end
c
c
c
       subroutine spectramkr(err,m,retil,f,snr,mb)
       real*8 err(m,m),retil(128),f(128),xii,xi,pi2
       real*8 sum(128)
       integer p
       pi2=6.283185307
       do 10 i=1,m
        sum(i)=0.0
10     continue
```

```
      do 20 i=1,m
       do 30 ii=1,m
        p=abs(i-ii)+1
        sum(p)=sum(p)+err(i,ii)
30     continue
20    continue
      print *, sum(1)
      do 50 i=1,128
       xi=real(i)/256
       retil(i)=0
       do 60 ii=1,m
        xii=ii-1
        retil(i)=retil(i)+sum(ii)*cos(pi2*xii*xi)
60     continue
50    continue
      do 80 i=1,128
       f(i)=real(i)/64
       retil(i)=retil(i)/(snr*mb)
80    continue
      return
      end
c
c
c
c     subroutine eigenmkr(m,gg,evec,eval,vec,val,g1,g2)
      subroutine eigenmkr(m,n,g,evec,eval,vec,val,g1,g2)
c     real*8 gg(m,m),evec(m,m),eval(m),temp(128),x
      real*8 g(m,m),evec(m,m),eval(m),temp(128),x
      real*8 vec(m,m),val(m),g1(m,m),g2(m,m)
      integer m,n,list(128),mid
c
c     call dmeigen(evec,eval,gg,m)
      call svdcmp(g,n,m,eval,evec)
        do 5 i=1,m
          eval(i)=eval(i)*eval(i)
5       continue
      x=-1d+308
      do 10 i=1,m
        temp(i)=eval(i)
10    continue
c ----------- Sort indices from largest eigenvalue to smallest ------
      do 20 i=1,m
        do 30 ii=1,m
          if(x.lt.temp(ii)) then
            x=temp(ii)
            list(i)=ii
          endif
30      continue
        temp(list(i))=0d0
        x=-1d+308
20    continue
c ----------- Order eigenvalues and eigenvectors --------------------
      do 40 i=1,m
        val(i)=eval(list(i))
        do 50 ii=1,m
          vec(ii,i)=evec(ii,list(i))
50      continue
40    continue
c ----------- Find middle of dynamic range --------------------------
      x=(log(val(m))+log(val(1)))/2d0
      do 60 i=1,m
        if(log(val(i)).lt.x) then
          mid=i
          goto 65
        endif
60    continue
c ----------- Generate two halves of inverse matrix -----------------
65    do 70 i=1,m
```

```
              do 80 ii=1,m
              g1(i,ii)=0d0
              g2(i,ii)=0d0
                do 90 iii=1,m
                  if(iii.lt.mid) then
                    g1(i,ii)=g1(i,ii)+
     *                       (1d0/val(iii))*vec(i,iii)*vec(ii,iii)
                  else
                    g2(i,ii)=g2(i,ii)+
     *                       (1d0/val(iii))*vec(i,iii)*vec(ii,iii)
                  endif
 90             continue
 80           continue
 70         continue
            return
            end
c
c
c
c     subroutine svdcmp(a,m,n,mp,np,w,v)
      subroutine svdcmp(a,m,n,w,v)
      implicit real*8 (a-h,o-z)
      parameter (nmax=128)
c     dimension a(mp,np),w(np),v(np,np),rv1(nmax)
      dimension a(m,n),w(n),v(n,n),rv1(nmax)
      g=0d0
      scale=0d0
      anorm=0d0
      do 25 i=1,n
        l=i+1
        rv1(i)=scale*g
        g=0d0
        s=0d0
        scale=0d0
        if (i.le.m) then
          do 11 k=i,m
            scale=scale+abs(a(k,i))
 11       continue
          if (scale.ne.0d0) then
            do 12 k=i,m
              a(k,i)=a(k,i)/scale
              s=s+a(k,i)*a(k,i)
 12         continue
            f=a(i,i)
            g=-sign(sqrt(s),f)
            h=f*g-s
            a(i,i)=f-g
            if (i.ne.n) then
              do 15 j=l,n
                s=0d0
                do 13 k=i,m
                  s=s+a(k,i)*a(k,j)
 13             continue
                f=s/h
                do 14 k=i,m
                  a(k,j)=a(k,j)+f*a(k,i)
 14             continue
 15           continue
            endif
            do 16 k= i,m
              a(k,i)=scale*a(k,i)
 16         continue
          endif
        endif
        w(i)=scale *g
        g=0d0
        s=0d0
        scale=0d0
```

```
          if ((i.le.m).and.(i.ne.n)) then
            do 17 k=l,n
              scale=scale+abs(a(i,k))
 17         continue
            if (scale.ne.0d0) then
              do 18 k=l,n
                a(i,k)=a(i,k)/scale
                s=s+a(i,k)*a(i,k)
 18           continue
              f=a(i,l)
              g=-sign(sqrt(s),f)
              h=f*g-s
              a(i,l)=f-g
              do 19 k=l,n
                rv1(k)=a(i,k)/h
 19           continue
              if (i.ne.m) then
                do 23 j=l,m
                  s=0d0
                  do 21 k=l,n
                    s=s+a(j,k)*a(i,k)
 21               continue
                  do 22 k=l,n
                    a(j,k)=a(j,k)+s*rv1(k)
 22               continue
 23             continue
              endif
              do 24 k=l,n
                a(i,k)=scale*a(i,k)
 24           continue
            endif
          endif
          anorm=max(anorm,(abs(w(i))+abs(rv1(i))))
 25     continue
        do 32 i=n,1,-1
          if (i.lt.n) then
            if (g.ne.0d0) then
              do 26 j=l,n
                v(j,i)=(a(i,j)/a(i,l))/g
 26           continue
              do 29 j=l,n
                s=0d0
                do 27 k=l,n
                  s=s+a(i,k)*v(k,j)
 27             continue
                do 28 k=l,n
                  v(k,j)=v(k,j)+s*v(k,i)
 28             continue
 29           continue
            endif
            do 31 j=l,n
              v(i,j)=0d0
              v(j,i)=0d0
 31         continue
          endif
          v(i,i)=1d0
          g=rv1(i)
          l=i
 32     continue
        do 39 i=n,1,-1
          l=i+1
          g=w(i)
          if (i.lt.n) then
            do 33 j=l,n
              a(i,j)=0d0
 33         continue
          endif
          if (g.ne.0d0) then
```

```
            g=1d0/g
            if (i.ne.n) then
              do 36 j=l,n
                s=0d0
                do 34 k=l,m
                  s=s+a(k,i)*a(k,j)
 34             continue
                f=(s/a(i,i))*g
                do 35 k=i,m
                  a(k,j)=a(k,j)+f*a(k,i)
 35             continue
 36           continue
            endif
            do 37 j=i,m
              a(j,i)=a(j,i)*g
 37         continue
          else
            do 38 j= i,m
              a(j,i)=0d0
 38         continue
          endif
          a(i,i)=a(i,i)+1d0
 39     continue
        do 49 k=n,1,-1
          do 48 its=1,30
            do 41 l=k,1,-1
              nm=l-1
              if ((abs(rv1(l))+anorm).eq.anorm)  go to 2
              if ((abs(w(nm))+anorm).eq.anorm)  go to 1
 41         continue
 1          c=0d0
            s=1d0
            do 43 i=l,k
              f=s*rv1(i)
              if ((abs(f)+anorm).ne.anorm) then
                g=w(i)
                h=sqrt(f*f+g*g)
                w(i)=h
                h=1d0/h
                c= (g*h)
                s=-(f*h)
                do 42 j=1,m
                  y=a(j,nm)
                  z=a(j,i)
                  a(j,nm)=(y*c)+(z*s)
                  a(j,i)=-(y*s)+(z*c)
 42             continue
              endif
 43         continue
 2          z=w(k)
            if (l.eq.k) then
              if (z.lt.0d0) then
                w(k)=-z
                do 44 j=1,n
                  v(j,k)=-v(j,k)
 44             continue
              endif
              go to 3
            endif
            if (its.eq.30) pause 'no convergence in 30 iterations'
            x=w(l)
            nm=k-1
            y=w(nm)
            g=rv1(nm)
            h=rv1(k)
            f=((y-z)*(y+z)+(g-h)*(g+h))/(2d0*h*y)
            g=sqrt(f*f+1d0)
            f=((x-z)*(x+z)+h*((y/(f+sign(g,f)))-h))/x
```

```
            c=1d0
            s=1d0
            do 47 j=l,nm
              i=j+1
              g=rv1(i)
              y=w(i)
              h=s*g
              g=c*g
              z=sqrt(f*f+h*h)
              rv1(j)=z
              c=f/z
              s=h/z
              f= (x*c)+(g*s)
              g=-(x*s)+(g*c)
              h=y*s
              y=y*c
              do 45 nm=1,n
                x=v(nm,j)
                z=v(nm,i)
                v(nm,j)= (x*c)+(z*s)
                v(nm,i)=-(x*s)+(z*c)
45            continue
              z=sqrt(f*f+h*h)
              w(j)=z
              if (z.ne.0d0) then
                z=1d0/z
                c=f*z
                s=h*z
              endif
              f= (c*g)+(s*y)
              x=-(s*g)+(c*y)
              do 46 nm=1,m
                y=a(nm,j)
                z=a(nm,i)
                a(nm,j)= (y*c)+(z*s)
                a(nm,i)=-(y*s)+(z*c)
46            continue
47          continue
            rv1(l)=0d0
            rv1(k)=f
            w(k)=x
48        continue
3         continue
49      continue
        return
        end
```

```
c
c
c     program gradsrch.f
c
c
      parameter(nn=638,mm=128)
      real*8 bw(nn,mm),h(mm,nn),retilu(128),f(128),x(mm),no(nn)
      real*8 v(mm,mm),w(mm),erru(mm,mm),retilc(128),err2(mm,mm)
      real*8 errc(mm,mm),u(nn,mm)
      real*8 y(nn),gg(mm,mm),d
      integer*4 random,window,cntr,ni,nj,nt,m,mb,n,l,k,nl,nl2
      character dfile*16
c
      window=3
      n=638
      l=128
      k=(n-1)/2.
      print *, 'nt?'
      read *, nt
      print *, 'ni?'
      read *, ni
      print *, 'nj?'
      read *, nj
      print *, 'mb?'
      read *, mb
      print *, 'm?'
      read *, m
      print *, 'snr?'
      read *, snr
      print *, 'd/D?'
      read *, d
      print *, 'Destination File?'
      read *, dfile
      nl=256
      nl2=nl/2
      call srandom(29)
      call bwmkr(bw,u,k,n,m,d,window)
      call ggmkr(bw,n,m,gg)
      call svdmkr(u,n,m,w,v)
      call hmkr(h,u,w,v,m,n)
      do 103 j=1,mm
       do 101 jj=1,mm
        erru(j,jj)=0
        errc(j,jj)=0
101    continue
103   continue
      cntr=0
      do 105 iv=1,nt
c       print *,iv
       if (float(iv)/10.0-int(float(iv)/10.0) .lt. 1e-10) then
        print *, iv
       endif
       call randm(x,no,m,n,snr,bw,y,mb)
       call grdsrch(x,h,no,erru,n,m,nt,errc,y,bw,gg,ni,nj,cntr)
105   continue
      print *, real(cntr)*real(nj)/real(nt)
      call spectramkr(erru,m,retilu,f,snr,mb)
      call spectramkr(errc,m,retilc,f,snr,mb)
      call plotfl(dfile,nl2,'d',f,'d',retilu)
      call plotfl(dfile,nl2,'d',f,'d',retilc)
      stop
      end
c
c
c
      subroutine bwmkr(bw,u,k,n,m,d,window)
      real*8 bw(n,m),u(n,m),hh,d
      integer*4 window
```

```
      do 500 i=1,n
       do 500 ii=1,m
500     bw(i,ii)=0d0
        u(i,ii)=0d0
      j=-k
      do 510 i=1,1+2*k
       bw(i,1)=hh(j,d,k,window)
       u(i,1)=bw(i,1)
       j=j+1
510   continue
      do 520 i=1,m-1
       do 530 ii=0,2*k
        bw(1+ii+i,i+1)=bw(ii+i,i)
        u(1+ii+i,i+1)=bw(ii+i,i)
530    continue
520   continue
      return
      end
c
c
c
      subroutine ggmkr(bw,n,m,gg)
      parameter(mm=128)
      real*8 bw(n,m),gg(m,m)
      do 600 i=1,m
       do 610 ii=1,m
        gg(i,ii)=0
        do 620 iii=1,n
         gg(i,ii)=gg(i,ii)+bw(iii,i)*bw(iii,ii)
         gg(i,ii)=g(i,ii)
620     continue
610    continue
600   continue
      return
      end
c
c
c
      subroutine randm(x,no,m,n,snr,bw,y,mb)
c
c       Generates random vectors x, no, and y=BWx+no.  x has mb
c       independent Rayleigh components with variance snr and m-mb
c       zero components.  no has n independent, zero mean, Gaussian
c       components with unit variance.
c
      real*8 u1,u2,u3,s
      real*8 x(m),no(n),bw(n,m),y(n)
      integer*4 MAXINTV,random
      parameter(MAXINTV=2147483647)
      n2=n/2
      do 500 i=1,n2
3      u1=real(random())/MAXINTV
       if(u1.gt.1.or.u1.eq.0)goto 3
2      u2=real(random())/MAXINTV
       if(u2.gt.1.or.u2.eq.0)goto 2
       no(2*i-1)=sqrt(-2*log(u1))*cos(6.2831853*u2)
       no(2*i)=sqrt(-2*log(u1))*sin(6.2831853*u2)
500   continue
      s=snr/2
      do 505 i=1,m
4      u3=real(random())/MAXINTV
       if(u3.gt.1.or.u3.eq.0)goto 4
       x(i)=sqrt(-2*log(u3))*sqrt(s)
505   continue
      do 507 i=mb+1,m
       x(i)=0
507   continue
      do 510 i=1,n
```

```
          y(i)=0
          do 520 ii=1,m
           y(i)=y(i)+bw(i,ii)*x(ii)
520       continue
           y(i)=y(i)+no(i)
510      continue
         return
         end
c
c
c
         subroutine grdsrch(x,h,no,erru,n,m,nt,errc,y,bw,gg,
     *              ni,nj,cntr)
         include 'fpedefs.f'
         parameter(mm=128)
         real*8 x(m),no(n),erru(m,m),h(m,n),eu(mm),zz,msq,msqold
         real*8 errc(m,m),gg(m,m),grad(mm),dmsq,s(mm)
         real*8 y(n),bw(n,m),z(mm),xg(mm)
         real*8 alpha,beta,ss,sggs,tem1,tem2
         integer cntr,nneg,list(mm),ni,nj,m,n,nt
         nneg=0
         msq=0
c ------------ Compute unconstrained solution and BWy --------------
         do 700 i=1,m
          xg(i)=0
          z(i)=0
          do 710 ii=1,n
           xg(i)=xg(i)+h(i,ii)*y(ii)
           z(i)=z(i)+bw(ii,i)*y(ii)
710       continue
          eu(i)=xg(i)-x(i)
          if(xg(i).lt.0)then
           nneg=nneg+1
          endif
700      continue
         if(nneg.eq.0) goto 200
c ---------------- Gradient Projection Algorithm -----------------------
c         ------ Disable overflow and divide-by-zero ---
c                floating point exceptions.
         nabls=fpgetxnabls()
         newabls=and(nabls,not(DIVO))
         newabls=and(newabls,not(OFLOW))
         call fpsetxnabls(newabls)
100      msqold=msq
         cntr=cntr+1
         do 770 iv=1,nj
          alpha=1.797d+308
          ss=0d0
          sggs=0d0
c      ------- Find active constraints and generate list  --------
          do 705 i=1,m
            list(i)=0
            if(xg(i).le.0d0) then
              xg(i)=0d0
              list(i)=1
            endif
705       continue
c      -------- Compute gradient and correction vector S  ---------
c               and pick step size alpha
          do 712 i=1,m
           grad(i)=0d0
           do 713 ii=1,m
             grad(i)=grad(i)+gg(i,ii)*xg(ii)
713        continue
           grad(i)=grad(i)-z(i)
c                ---- Project gradient onto active,    ----
c                     nonobstructing constraints.
           if(list(i).eq.1.and.grad(i).gt.0d0) then
```

```
                s(i)=0d0
              else
                s(i)=-grad(i)
              endif
c       -----  Determine step size alpha:          --------------
c               Determine minimum step size for next constraint,
c               Determine step size for minimum along direction S,
c       -----  Pick smaller of the two       -----------------------
              if(list(i).ne.1) then
                beta=xg(i)/grad(i)
                if(beta.gt.0d0.and.beta.lt.alpha) then
                  alpha=beta
                endif
              endif
712         continue
            do 714 i=1,m
              ss=ss+s(i)*s(i)
              do 716 j=1,m
               sggs=sggs+s(i)*gg(i,j)*s(j)
716           continue
714         continue
            beta=ss/sggs
            if(beta.gt.0d0.and.beta.lt.alpha) then
              alpha=beta
c ------------compute new gradient and test for orthogonality ----------
c             with the S direction.
c               zz=0d0
c               do 900 i=1,m
c                grad(i)=0d0
c                do 901 ii=1,m
c                  grad(i)=grad(i)+gg(i,ii)*(xg(ii)+alpha*s(ii))
c901              continue
c                grad(i)=grad(i)-z(i)
c                zz=zz+grad(i)*s(i)
c900             continue
c               print *, 'gTs =', zz
c ------------ End of Test -------------------------------------------
            endif
c       -----------  Compute new Solution   ------
            do 740 i=1,m
              xg(i)=xg(i)+alpha*s(i)
740         continue
770        continue
c ------------------------ Check ||y-Gx|| squared -----------------------
           msq=0
           do 780 i=1,n
            zz=0
            do 790 ii=1,m
             zz=zz+bw(i,ii)*xg(ii)
790         continue
            zz=y(i)-zz
            msq=msq+zz*zz
780        continue
c              print *, msq
           dmsq=abs((msqold-msq)*(10**ni))
           if (dmsq.gt.1e+9) goto 100
           if (int(dmsq).ne.0) goto 100
c          -----  Enable Floating Point Exceptions  ---
            call fpsetxflags(0)
            call fpsetxnabls(nabls)
c ---------  Accumulate Error Vector Outer Product and Return ------------
200        do 720 i=1,m
              print *, xg(i), grad(i), s(i)
            tem1=xg(i)-x(i)
            do 730 ii=1,m
             tem2=xg(ii)-x(ii)
             erru(i,ii)=erru(i,ii)+eu(i)*eu(ii)/nt
             errc(i,ii)=errc(i,ii)+tem1*tem2/nt
```

```
730      continue
720     continue
        print *, msq
        return
        end
c
c
c
        subroutine spectramkr(err,m,retil,f,snr,mb)
        real*8 err(m,m),retil(128),f(128),xii,xi,pi2,sum(128)
        integer p
        pi2=6.283185307
        do 10 i=1,m
         sum(i)=0.0
10      continue
        do 20 i=1,m
         do 30 ii=1,m
          p=abs(i-ii)+1
          sum(p)=sum(p)+err(i,ii)
30       continue
20      continue
        print *, sum(1)
        do 50 i=1,128
         xi=real(i)/256
         retil(i)=0
         do 60 ii=1,m
          xii=ii-1
          retil(i)=retil(i)+sum(ii)*cos(pi2*xii*xi)
60       continue
50      continue
        do 80 i=1,128
         f(i)=real(i)/64
         retil(i)=retil(i)/(snr*mb)
80      continue
        return
        end
c
c
c
        function hh(j,d,k,window)
        real*8 hh,d,k1,pi,pi2,j1
        integer*4  k,j,window
        pi=3.141592654
        pi2=2*pi
        j1=j
        k1=k
        if(j.eq.0)then
         hh=d*d
        else
         hh=cos(.78539816*dble(j)*(1-d))*sin(.78539816*d*dble(j))/
     c   (.78539816*dble(j))
         hh=hh*hh
        endif
        if (window.eq.1) then
          hh=hh
         elseif (window.eq.2) then
          hh=hh*(1-abs(j1)/k1)
         elseif (window.eq.3) then
          hh=hh*(.5+.5*cos(pi*j1/k1))
         elseif (window.eq.4) then
          hh=hh*(0.54+0.46*cos(pi*j1/k1))
         else
          hh=hh*(0.42+0.5*cos(pi*j1/k1)+0.08*cos(pi2*j1/k1))
        endif
10      return
        end
c
c
```

```
c
         subroutine hmkr(h,u,w,v,m,n)
c
         real*8 u(m,n),w(n),v(n,n),h(n,m)
         integer m,n
c
         do 10 i=1,n
           do 20 j=1,m
             do 30 k=1,n
               h(i,j)=(1/w(k))*v(i,k)*u(j,k)
30           continue
20         continue
10       continue
c
c
c
      subroutine svdmkr(a,m,n,w,v)
c
c        This routine generates the SVD of an mxn matrix A
c        where A = UWVtranspose, with U mxn, and V and W are
c        nxn.  U is column orthogonal, V is row and column
c        orthogonal, and W is diagonal.  U is returned in the
c        array a.  The diagonal of W is returned as the vector w.
c
      implicit real*8 (a-h,o-z)
      parameter (nmax=128)
      dimension a(m,n),w(n),v(n,n),rv1(nmax)
c
      g=0d0
      scale=0d0
      anorm=0d0
      do 25 i=1,n
        l=i+1
        rv1(i)=scale*g
        g=0d0
        s=0d0
        scale=0d0
        if (i.le.m) then
          do 11 k=i,m
            scale=scale+abs(a(k,i))
11        continue
          if (scale.ne.0d0) then
            do 12 k=i,m
              a(k,i)=a(k,i)/scale
              s=s+a(k,i)*a(k,i)
12          continue
            f=a(i,i)
            g=-sign(sqrt(s),f)
            h=f*g-s
            a(i,i)=f-g
            if (i.ne.n) then
              do 15 j=l,n
                s=0d0
                do 13 k=i,m
                  s=s+a(k,i)*a(k,j)
13              continue
                f=s/h
                do 14 k=i,m
                  a(k,j)=a(k,j)+f*a(k,i)
14              continue
15            continue
            endif
            do 16 k= i,m
              a(k,i)=scale*a(k,i)
16          continue
          endif
        endif
        w(i)=scale *g
```

```
        g=0d0
        s=0d0
        scale=0d0
        if ((i.le.m).and.(i.ne.n)) then
          do 17 k=1,n
            scale=scale+abs(a(i,k))
17        continue
          if (scale.ne.0d0) then
            do 18 k=1,n
              a(i,k)=a(i,k)/scale
              s=s+a(i,k)*a(i,k)
18          continue
            f=a(i,l)
            g=-sign(sqrt(s),f)
            h=f*g-s
            a(i,l)=f-g
            do 19 k=l,n
              rv1(k)=a(i,k)/h
19          continue
            if (i.ne.m) then
              do 23 j=l,m
                s=0d0
                do 21 k=l,n
                  s=s+a(j,k)*a(i,k)
21              continue
                do 22 k=l,n
                  a(j,k)=a(j,k)+s*rv1(k)
22              continue
23            continue
            endif
            do 24 k=l,n
              a(i,k)=scale*a(i,k)
24          continue
          endif
        endif
        anorm=max(anorm,(abs(w(i))+abs(rv1(i))))
25    continue
      do 32 i=n,1,-1
        if (i.lt.n) then
          if (g.ne.0d0) then
            do 26 j=l,n
              v(j,i)=(a(i,j)/a(i,l))/g
26          continue
            do 29 j=l,n
              s=0d0
              do 27 k=l,n
                s=s+a(i,k)*v(k,j)
27            continue
              do 28 k=l,n
                v(k,j)=v(k,j)+s*v(k,i)
28            continue
29          continue
          endif
          do 31 j=l,n
            v(i,j)=0d0
            v(j,i)=0d0
31        continue
        endif
        v(i,i)=1d0
        g=rv1(i)
        l=i
32    continue
      do 39 i=n,1,-1
        l=i+1
        g=w(i)
        if (i.lt.n) then
          do 33 j=l,n
            a(i,j)=0d0
```

```
33        continue
        endif
        if (g.ne.0d0) then
          g=1d0/g
          if (i.ne.n) then
            do 36 j=l,n
              s=0d0
              do 34 k=l,m
                s=s+a(k,i)*a(k,j)
34            continue
              f=(s/a(i,i))*g
              do 35 k=i,m
                a(k,j)=a(k,j)+f*a(k,i)
35            continue
36          continue
          endif
          do 37 j=i,m
            a(j,i)=a(j,i)*g
37        continue
        else
          do 38 j= i,m
            a(j,i)=0d0
38        continue
        endif
        a(i,i)=a(i,i)+1d0
39    continue
      do 49 k=n,1,-1
        do 48 its=1,30
          do 41 l=k,1,-1
            nm=l-1
            if ((abs(rv1(l))+anorm).eq.anorm)  go to 2
            if ((abs(w(nm))+anorm).eq.anorm)  go to 1
41        continue
1         c=0d0
          s=1d0
          do 43 i=l,k
            f=s*rv1(i)
            if ((abs(f)+anorm).ne.anorm) then
              g=w(i)
              h=sqrt(f*f+g*g)
              w(i)=h
              h=1d0/h
              c= (g*h)
              s=-(f*h)
              do 42 j=1,m
                y=a(j,nm)
                z=a(j,i)
                a(j,nm)=(y*c)+(z*s)
                a(j,i)=-(y*s)+(z*c)
42            continue
            endif
43        continue
2         z=w(k)
          if (l.eq.k) then
            if (z.lt.0d0) then
              w(k)=-z
              do 44 j=1,n
                v(j,k)=-v(j,k)
44            continue
            endif
            go to 3
          endif
          if (its.eq.30) pause 'no convergence in 30 iterations'
          x=w(l)
          nm=k-1
          y=w(nm)
          g=rv1(nm)
          h=rv1(k)
```

```
          f=((y-z)*(y+z)+(g-h)*(g+h))/(2d0*h*y)
          g=sqrt(f*f+1d0)
          f=((x-z)*(x+z)+h*((y/(f+sign(g,f)))-h))/x
          c=1d0
          s=1d0
          do 47 j=l,nm
            i=j+1
            g=rv1(i)
            y=w(i)
            h=s*g
            g=c*g
            z=sqrt(f*f+h*h)
            rv1(j)=z
            c=f/z
            s=h/z
            f= (x*c)+(g*s)
            g=-(x*s)+(g*c)
            h=y*s
            y=y*c
            do 45 nm=1,n
              x=v(nm,j)
              z=v(nm,i)
              v(nm,j)= (x*c)+(z*s)
              v(nm,i)=-(x*s)+(z*c)
45          continue
            z=sqrt(f*f+h*h)
            w(j)=z
            if (z.ne.0d0) then
              z=1d0/z
              c=f*z
              s=h*z
            endif
            f= (c*g)+(s*y)
            x=-(s*g)+(c*y)
            do 46 nm=1,m
              y=a(nm,j)
              z=a(nm,i)
              a(nm,j)= (y*c)+(z*s)
              a(nm,i)=-(y*s)+(z*c)
46          continue
47        continue
          rv1(l)=0d0
          rv1(k)=f
          w(k)=x
48      continue
3       continue
49    continue
      return
      end
```

```
c
c
c          program minvar.f
c
c
           parameter(mm=128)
           real*8 bw(1150,mm),retil(128),f(128)
           real*8 g(mm,mm),d,snr
           real*4 eps
           integer*4 m,window,k,l,n,im,i,ii,ier
           character*16 dfile
c
           print *, 'k?'
           read *, k
           print *, 'Window number?'
           read *, window
           print *, 'Destination File?'
           read *, dfile
           l=128
           n=2*k+1
           eps=1e-6
           do 90 im=1,6
            m=2**(im+1)
            do 100 i=1,5
             d=dble(i)/1d+1
             call bwmkr(bw,k,n,m,d,window)
             do 110 ii=2,4
              snr=10**ii
              call two(bw,n,m,g,snr,ier,eps)
              call spectramkr(g,m,retil,f)
              call plotfl(dfile,128,'d',f,'d',retil)
  110        continue
  100       continue
   90      continue
           stop
           end
c
c
c
           subroutine two(bw,n,m,g,snr,ier,eps)
c           subroutine to calculate h=(gtg)*snr + i inv
           parameter(mm=128)
           real*8 bw(n,m),g(m,m),r(8256),snr
           real*4 eps
           integer ier
           do 600 i=1,m
            do 610 ii=1,m
             g(i,ii)=0
             do 620 iii=1,n
              g(i,ii)=g(i,ii)+bw(iii,i)*bw(iii,ii)*snr
  620        continue
            if(i.eq.ii)g(i,ii)=g(i,ii)+1d+0
  610       continue
  600      continue
           call squartri(r,m,g)
           call dsinv(r,m,eps,ier)
           call trisquar(g,m,r)
           return
           end
c
c
c
           subroutine bwmkr(bw,k,n,m,d,window)
           real*8 bw(n,m),hh,d
           integer n,m,k,window
           do 500 i=1,n
            do 500 ii=1,m
  500        bw(i,ii)=0
```

```
      j=-k
      do 510 i=1,1+2*k
       bw(i,1)=hh(j,d,k,window)
       j=j+1
510   continue
      do 520 i=1,m-1
       do 530 ii=0,2*k
        bw(1+ii+i,i+1)=bw(ii+i,i)
530    continue
520   continue
      return
      end
c
c
c
      function hh(j,d,k,window)
      real*8 hh,d,k1,pi,pi2,j1
      integer k,j,m,window
      m=window
      pi=3.141592654
      pi2=2*pi
      j1=j
      k1=k
      if(j.eq.0)then
       hh=d*d
      else
       hh=cos(.78539816*dble(j)*(1-d))*sin(.78539816*d*dble(j))/
     c (.78539816*dble(j))
       hh=hh*hh
      endif
      if (m.eq.1) then
        hh=hh
       elseif (m.eq.2) then
        hh=hh*(1-abs(j1)/k1)
       elseif (m.eq.3) then
        hh=hh*(.5+.5*cos(pi*j1/k1))
       elseif (m.eq.4) then
        hh=hh*(0.54+0.46*cos(pi*j1/k1))
       else
        hh=hh*(0.42+0.5*cos(pi*j1/k1)+0.08*cos(pi2*j1/k1))
      endif
10    return
      end
c
c
c
      subroutine spectramkr(err,m,retil,f)
      real*8 err(m,m),retil(128),f(128),xii,xi,pi2,sum(128)
      integer p
      pi2=6.283185307
      do 10 i=1,m
       sum(i)=0.0
10    continue
      do 20 i=1,m
       do 30 ii=1,m
        p=abs(i-ii)+1
        sum(p)=sum(p)+err(i,ii)
30     continue
20    continue
      do 50 i=1,128
       xi=real(i)/256
       retil(i)=0
       do 60 ii=1,m
        xii=ii-1
        retil(i)=retil(i)+sum(ii)*cos(pi2*xii*xi)
60     continue
50    continue
      do 80 i=1,128
```

```
       f(i)=real(i)/64
       retil(i)=retil(i)/dble(m)
 80   continue
      return
      end
```

```
c
c
c            program snrmkr.f
c
c
      real*4 sum,snr(128),f1(128),lg10
      real*8 f(128),e(128)
      character*12 file1
      character*14 file2
      file1='m16mb16d1w5.'
      file2='snrm16mb16d1w5'
      call readfl(file1//'2',n,'d',f,'d',e,1,1,128)
      lg10=log(1e1)
      sum=0
      do 10 i=1,128
       sum=sum+e(i)
c      snr(i)=real(i)/sum
       snr(i)=10*log(real(i)/sum)/lg10
       f1(i)=real(i)/64
 10   continue
      call plotfl(file2,128,'f',f1,'f',snr)
      call readfl(file1//'2',n,'d',f,'d',e,3,1,128)
      sum=0
      do 20 i=1,128
       sum=sum+e(i)
c      snr(i)=real(i)/sum
       snr(i)=10*log(real(i)/sum)/lg10
 20   continue
      call plotfl(file2,128,'f',f1,'f',snr)
      call readfl(file1//'3',n,'d',f,'d',e,1,1,128)
      sum=0
      do 30 i=1,128
       sum=sum+e(i)
c      snr(i)=real(i)/sum
       snr(i)=10*log(real(i)/sum)/lg10
 30   continue
      call plotfl(file2,128,'f',f1,'f',snr)
      call readfl(file1//'3',n,'d',f,'d',e,3,1,128)
      sum=0
      do 40 i=1,128
       sum=sum+e(i)
c      snr(i)=real(i)/sum
       snr(i)=10*log(real(i)/sum)/lg10
 40   continue
      call plotfl(file2,128,'f',f1,'f',snr)
      call readfl(file1//'4',n,'d',f,'d',e,1,1,128)
      sum=0
      do 50 i=1,128
       sum=sum+e(i)
c      snr(i)=real(i)/sum
       snr(i)=10*log(real(i)/sum)/lg10
 50   continue
      call plotfl(file2,128,'f',f1,'f',snr)
      call readfl(file1//'4',n,'d',f,'d',e,3,1,128)
      sum=0
      do 60 i=1,128
       sum=sum+e(i)
c      snr(i)=real(i)/sum
       snr(i)=10*log(real(i)/sum)/lg10
 60   continue
      call plotfl(file2,128,'f',f1,'f',snr)
      call readfl(file1//'5',n,'d',f,'d',e,1,1,128)
      sum=0
      do 70 i=1,128
       sum=sum+e(i)
c      snr(i)=real(i)/sum
       snr(i)=10*log(real(i)/sum)/lg10
       f1(i)=real(i)/64
```

```
   70 continue
      call plotfl(file2,128,'f',f1,'f',snr)
      call readfl(file1//'5',n,'d',f,'d',e,3,1,128)
      sum=0
      do 80 i=1,128
       sum=sum+e(i)
c      snr(i)=real(i)/sum
       snr(i)=10*log(real(i)/sum)/lg10
   80 continue
      call plotfl(file2,128,'f',f1,'f',snr)
      call readfl(file1//'6',n,'d',f,'d',e,1,1,128)
      sum=0
      do 90 i=1,128
       sum=sum+e(i)
c      snr(i)=real(i)/sum
       snr(i)=10*log(real(i)/sum)/lg10
   90 continue
      call plotfl(file2,128,'f',f1,'f',snr)
      call readfl(file1//'6',n,'d',f,'d',e,3,1,128)
      sum=0
      do 100 i=1,128
       sum=sum+e(i)
c      snr(i)=real(i)/sum
       snr(i)=10*log(real(i)/sum)/lg10
  100 continue
      call plotfl(file2,128,'f',f1,'f',snr)
      call readfl(file1//'7',n,'d',f,'d',e,1,1,128)
      sum=0
      do 110 i=1,128
       sum=sum+e(i)
c      snr(i)=real(i)/sum
       snr(i)=10*log(real(i)/sum)/lg10
  110 continue
      call plotfl(file2,128,'f',f1,'f',snr)
      call readfl(file1//'7',n,'d',f,'d',e,3,1,128)
      sum=0
      do 120 i=1,128
       sum=sum+e(i)
c      snr(i)=real(i)/sum
       snr(i)=10*log(real(i)/sum)/lg10
  120 continue
      call plotfl(file2,128,'f',f1,'f',snr)
      call readfl(file1//'8',n,'d',f,'d',e,1,1,128)
      sum=0
      do 130 i=1,128
       sum=sum+e(i)
c      snr(i)=real(i)/sum
       snr(i)=10*log(real(i)/sum)/lg10
  130 continue
      call plotfl(file2,128,'f',f1,'f',snr)
      call readfl(file1//'8',n,'d',f,'d',e,3,1,128)
      sum=0
      do 140 i=1,128
       sum=sum+e(i)
c      snr(i)=real(i)/sum
       snr(i)=10*log(real(i)/sum)/lg10
  140 continue
      call plotfl(file2,128,'f',f1,'f',snr)
      stop
      end
```

```
c
c
c       routine svd.f
c
c
c       This routine generates the SVD decomposition of an
c       mxn matrix A = UWVtranspose, where U is mxn, and V
c       and W are nxn.  U is column orthogonal, V is row and
c       column orthogonal, and W is diagonal.  U is returned
c       in the array a.
c
c
c
      subroutine svdcmp(a,m,n,w,v)
      implicit real*8 (a-h,o-z)
      parameter (nmax=128)
      dimension a(m,n),w(n),v(n,n),rv1(nmax)
c
      g=0d0
      scale=0d0
      anorm=0d0
      do 25 i=1,n
        l=i+1
        rv1(i)=scale*g
        g=0d0
        s=0d0
        scale=0d0
        if (i.le.m) then
          do 11 k=i,m
            scale=scale+abs(a(k,i))
 11       continue
          if (scale.ne.0d0) then
            do 12 k=i,m
              a(k,i)=a(k,i)/scale
              s=s+a(k,i)*a(k,i)
 12         continue
            f=a(i,i)
            g=-sign(sqrt(s),f)
            h=f*g-s
            a(i,i)=f-g
            if (i.ne.n) then
              do 15 j=l,n
                s=0d0
                do 13 k=i,m
                  s=s+a(k,i)*a(k,j)
 13             continue
                f=s/h
                do 14 k=i,m
                  a(k,j)=a(k,j)+f*a(k,i)
 14             continue
 15           continue
            endif
            do 16 k= i,m
              a(k,i)=scale*a(k,i)
 16         continue
          endif
        endif
        w(i)=scale *g
        g=0d0
        s=0d0
        scale=0d0
        if ((i.le.m).and.(i.ne.n)) then
          do 17 k=l,n
            scale=scale+abs(a(i,k))
 17       continue
          if (scale.ne.0d0) then
            do 18 k=l,n
              a(i,k)=a(i,k)/scale
```

```
                s=s+a(i,k)*a(i,k)
18            continue
              f=a(i,l)
              g=-sign(sqrt(s),f)
              h=f*g-s
              a(i,l)=f-g
              do 19 k=l,n
                rv1(k)=a(i,k)/h
19            continue
              if (i.ne.m) then
                do 23 j=l,m
                  s=0d0
                  do 21 k=l,n
                    s=s+a(j,k)*a(i,k)
21                continue
                  do 22 k=l,n
                    a(j,k)=a(j,k)+s*rv1(k)
22                continue
23              continue
              endif
              do 24 k=l,n
                a(i,k)=scale*a(i,k)
24            continue
            endif
          endif
          anorm=max(anorm,(abs(w(i))+abs(rv1(i))))
25      continue
        do 32 i=n,1,-1
          if (i.lt.n) then
            if (g.ne.0d0) then
              do 26 j=l,n
                v(j,i)=(a(i,j)/a(i,l))/g
26            continue
              do 29 j=l,n
                s=0d0
                do 27 k=l,n
                  s=s+a(i,k)*v(k,j)
27              continue
                do 28 k=l,n
                  v(k,j)=v(k,j)+s*v(k,i)
28              continue
29            continue
            endif
            do 31 j=l,n
              v(i,j)=0d0
              v(j,i)=0d0
31          continue
          endif
          v(i,i)=1d0
          g=rv1(i)
          l=i
32      continue
        do 39 i=n,1,-1
          l=i+1
          g=w(i)
          if (i.lt.n) then
            do 33 j=l,n
              a(i,j)=0d0
33          continue
          endif
          if (g.ne.0d0) then
            g=1d0/g
            if (i.ne.n) then
              do 36 j=l,n
                s=0d0
                do 34 k=l,m
                  s=s+a(k,i)*a(k,j)
34              continue
```

```
                f=(s/a(i,i))*g
                do 35 k=i,m
                  a(k,j)=a(k,j)+f*a(k,i)
35              continue
36            continue
            endif
            do 37 j=i,m
              a(j,i)=a(j,i)*g
37          continue
          else
            do 38 j= i,m
              a(j,i)=0d0
38          continue
          endif
          a(i,i)=a(i,i)+1d0
39      continue
        do 49 k=n,1,-1
          do 48 its=1,30
            do 41 l=k,1,-1
              nm=l-1
              if ((abs(rv1(l))+anorm).eq.anorm)  go to 2
              if ((abs(w(nm))+anorm).eq.anorm)  go to 1
41          continue
1           c=0d0
            s=1d0
            do 43 i=l,k
              f=s*rv1(i)
              if ((abs(f)+anorm).ne.anorm) then
                g=w(i)
                h=sqrt(f*f+g*g)
                w(i)=h
                h=1d0/h
                c= (g*h)
                s=-(f*h)
                do 42 j=1,m
                  y=a(j,nm)
                  z=a(j,i)
                  a(j,nm)=(y*c)+(z*s)
                  a(j,i)=-(y*s)+(z*c)
42              continue
              endif
43          continue
2           z=w(k)
            if (l.eq.k) then
              if (z.lt.0d0) then
                w(k)=-z
                do 44 j=1,n
                  v(j,k)=-v(j,k)
44              continue
              endif
              go to 3
            endif
            if (its.eq.30) pause 'no convergence in 30 iterations'
            x=w(l)
            nm=k-1
            y=w(nm)
            g=rv1(nm)
            h=rv1(k)
            f=((y-z)*(y+z)+(g-h)*(g+h))/(2d0*h*y)
            g=sqrt(f*f+1d0)
            f=((x-z)*(x+z)+h*((y/(f+sign(g,f)))-h))/x
            c=1d0
            s=1d0
            do 47 j=l,nm
              i=j+1
              g=rv1(i)
              y=w(i)
              h=s*g
```

```
            g=c*g
            z=sqrt(f*f+h*h)
            rv1(j)=z
            c=f/z
            s=h/z
            f= (x*c)+(g*s)
            g=-(x*s)+(g*c)
            h=y*s
            y=y*c
            do 45 nm=1,n
              x=v(nm,j)
              z=v(nm,i)
              v(nm,j)= (x*c)+(z*s)
              v(nm,i)=-(x*s)+(z*c)
45          continue
            z=sqrt(f*f+h*h)
            w(j)=z
            if (z.ne.0d0) then
              z=1d0/z
              c=f*z
              s=h*z
            endif
            f= (c*g)+(s*y)
            x=-(s*g)+(c*y)
            do 46 nm=1,m
              y=a(nm,j)
              z=a(nm,i)
              a(nm,j)= (y*c)+(z*s)
              a(nm,i)=-(y*s)+(z*c)
46          continue
47        continue
          rv1(l)=0d0
          rv1(k)=f
          w(k)=x
48      continue
3       continue
49    continue
      return
      end
```

```
c
c
c       program minvar2.f
c
c
        parameter(mm=128)
        real*8 bw(1150,mm),retil(128),f(128)
        real*8 g(mm,mm),d,snr
        real*4 eps
        integer*4 m,n,im,i,ii,ni,ier,max,uniform,apertures
        character*16 dfile
c
        n=1024
        print *, 'snr?'
        read *, snr
        print *, 'd/D?'
        read *, d
        print *, 'uniformity?(yes=1,no=0)'
        read *, uniform
        print *, 'number of apertures?'
        read *, apertures
        print *, 'Largest support?'
        read *, max
        print *, 'Destination File?'
        read *, dfile
        eps=1e-6
        do 90 im=1,max
          m=4*im
          call bwmkr(bw,n,m,d,uniform,apertures)
          call two(bw,n,m,g,snr,ier,eps)
          call spectramkr(g,m,retil,f)
          call plotfl(dfile,128,'d',f,'d',retil)
90      continue
        stop
        end
c
c
c
        subroutine two(bw,n,m,g,snr,ier,eps)
c        subroutine to calculate h=(gtg)*snr + i inv
        parameter(mm=128)
        real*8 bw(n,m),g(m,m),r(8256),snr
        real*4 eps
        integer ier
        do 600 i=1,m
         do 610 ii=1,m
          g(i,ii)=0
          do 620 iii=1,n
           g(i,ii)=g(i,ii)+bw(iii,i)*bw(iii,ii)*snr
620       continue
         if(i.eq.ii)g(i,ii)=g(i,ii)+1d+0
610      continue
600     continue
        call squartri(r,m,g)
        call dsinv(r,m,eps,ier)
        call trisquar(g,m,r)
        return
        end
c
c
c
        subroutine bwmkr(bw,n,m,d,uniform,apertures)
        real*8 bw(n,m),hh,d
        integer n,m,uniform,apertures
        do 520 i=1,n
         do 530 ii=1,m
          bw(i,ii)=hh((m-n)/2-ii+i,d,uniform,apertures)
530      continue
```

```
 520        continue
            return
            end
c
c
c
        function hh(j,d,uniform,apertures)
        real*8 hh,d,pi2,d1,j1,j2,x1,x2,xj,xk,xi
        integer j,k,l,uniform,apertures
c
        k=apertures
        xk=k
        if(j.eq.0)then
          hh=xk*d/4d0
        else
          pi2=2d0*atan(1d0)
          d1=1d0-d
          xj=j
          j1=xj*d1*pi2
          j2=xj*d*pi2/2d0
          x1=sin(j2)/j2
          x1=x1*x1*d/4d0
          x2=0d0
          if(uniform.eq.1) then
              do 10 i=1,k-1
                xi=i
                x2=x2+(1d0-xi/xk)*cos(xi*j1/(xk-1d0))
 10           continue
              hh=x1*(1d0+2d0*x2)
            elseif(uniform.eq.0.and.apertures.eq.3) then
              do 20 i=1,3
                xi=i
                x2=x2+cos(xi*j1/3d0)
 20           continue
              hh=x1*(1d0+2d0*x2/3d0)
            elseif(uniform.eq.0.and.apertures.eq.4) then
              do 30 i=1,6
                xi=i
                x2=x2+cos(xi*j1/6d0)
 30           continue
              hh=x1*(1d0+x2/2d0)
            elseif(uniform.eq.0.and.apertures.eq.5) then
              do 40 i=1,11
                if(i.eq.10) goto 40
                xi=i
                x2=x2+cos(xi*j1/11d0)
 40           continue
              hh=x1*(1d0+2d0*x2/5d0)
            elseif(uniform.eq.0.and.apertures.eq.6) then
              do 50 i=1,17
                if(i.eq.14.or.i.eq.15) goto 50
                xi=i
                x2=x2+cos(xi*j1/17d0)
 50           continue
              hh=x1*(1d0+x2/3d0)
          endif
        endif
        return
        end
c
c
c
            subroutine spectramkr(err,m,retil,f)
            real*8 err(m,m),retil(128),f(128),xii,xi,pi2,sum(128)
            integer p
            pi2=6.283185307
            do 10 i=1,m
              sum(i)=0.0
```

```
10    continue
      do 20 i=1,m
       do 30 ii=1,m
        p=abs(i-ii)+1
        sum(p)=sum(p)+err(i,ii)
30     continue
20    continue
      do 50 i=1,128
       xi=real(i)/256
       retil(i)=0
       do 60 ii=1,m
        xii=ii-1
        retil(i)=retil(i)+sum(ii)*cos(pi2*xii*xi)
60     continue
50    continue
      do 80 i=1,128
       f(i)=real(i)/64
       retil(i)=retil(i)/dble(m)
80    continue
      return
      end
```

```
c
c
c          program snrmkr2 f
c
c
      real*4 sum,snr(128),f1(128),lg10,slice(32),count(32)
      real*8 f(128),e(128)
c
      lg10=log(1e1)
c     do 5 j=1,180
c     do 5 j=1,8
      do 5 j=1,32
c       call readfl('minvar2n1024',n,'d',f,'d',e,j,1,128)
c       call readfl('newmtfd005s6',n,'d',f,'d',e,j,1,128)
        call readfl('s5d005u0a6s32',n,'d',f,'d',e,j,1,128)
c       call readfl('s4d005u1a11s32',n,'d',f,'d',e,j,1,128)
c       call readfl('s6d005u1a6s8',n,'d',f,'d',e,j,1,128)
        sum=0
        do 10 i=1,128
          sum=sum+e(i)
          snr(i)=10*log(real(i)/sum)/lg10
          f1(i)=real(i)/64
10      continue
c       call plotfl('snrminvarn1024',128,'f',f1,'f',snr)
c       call plotfl('snrnwmtfd005s6',128,'f',f1,'f',snr)
        call plotfl('snrs5d005u0a6',128,'f',f1,'f',snr)
c       call plotfl('snrs4d005u1a11',128,'f',f1,'f',snr)
c       slice(j)=snr(64)
c       slice(j)=snr(52)
        slice(j)=snr(49)
        count(j)=j
5     continue
      call plotfl('slcs5d005u0a6',32,'f',count,'f',slice)
c     call plotfl('slcs4d005u1a11',32,'f',count,'f',slice)
c     call plotfl('slcs6d005u1a6',8,'f',count,'f',slice)
      stop
      end
```

APPENDIX C for Chapter 4

Computer Listings

This appendix contains the Fortran source code of the computer programs used to generate the data for this chapter.

```
c
c
c         program minvar2.f
c
c
          parameter(mm=128)
          real*8 bw(1150,mm),retil(128),f(128)
          real*8 g(mm,mm),d,snr
          real*4 eps
          integer*4 m,n,im,i,ii,ni,ier,max,uniform,apertures
          character*16 dfile
c
          n=1024
          print *, 'snr?'
          read *, snr
          print *, 'd/D?'
          read *, d
          print *, 'uniformity?(yes=1,no=0)'
          read *, uniform
          print *, 'number of apertures?'
          read *, apertures
          print *, 'Largest support?'
          read *, max
          print *, 'Destination File?'
          read *, dfile
          eps=1e-6
          do 90 im=1,max
            m=4*im
            call bwmkr(bw,n,m,d,uniform,apertures)
            call two(bw,n,m,g,snr,ier,eps)
            call spectramkr(g,m,retil,f)
            call plotfl(dfile,128,'d',f,'d',retil)
 90       continue
          stop
          end
c
c
c
          subroutine two(bw,n,m,g,snr,ier,eps)
c          subroutine to calculate h=(gtg)*snr + i inv
          parameter(mm=128)
          real*8 bw(n,m),g(m,m),r(8256),snr
          real*4 eps
          integer ier
          do 600 i=1,m
           do 610 ii=1,m
            g(i,ii)=0
            do 620 iii=1,m
             g(i,ii)=g(i,ii)+bw(iii,i)*bw(iii,ii)*snr
 620        continue
           if(i.eq.ii)g(i,ii)=g(i,ii)+1d+0
 610       continue
 600      continue
          call squartri(r,m,g)
          call dsinv(r,m,eps,ier)
          call trisquar(g,m,r)
          return
          end
c
```

```
c
c
            subroutine bwmkr(bw,n,m,d,uniform,apertures)
            real*8 bw(n,m),hh,d
            integer n,m,uniform,apertures
            do 520 i=1,n
             do 530 ii=1,m
              bw(i,ii)=hh((m-n)/2-ii+i,d,uniform,apertures)
530          continue
520         continue
            return
            end
c
c
c
      function hh(j,d,uniform,apertures)
      real*8 hh,d,pi2,d1,j1,j2,x1,x2,xj,xk,xi
      integer j,k,l,uniform,apertures
c
      k=apertures
      xk=k
      if(j.eq.0)then
        hh=xk*d/4d0
      else
        pi2=2d0*atan(1d0)
        d1=1d0-d
        xj=j
        j1=xj*d1*pi2
        j2=xj*d*pi2/2d0
        x1=sin(j2)/j2
        x1=x1*x1*d/4d0
        x2=0d0
        if(uniform.eq.1) then
            do 10 i=1,k-1
              xi=i
              x2=x2+(1d0-xi/xk)*cos(xi*j1/(xk-1d0))
10          continue
            hh=x1*(1d0+2d0*x2)
          elseif(uniform.eq.0.and.apertures.eq.3) then
            do 20 i=1,3
              xi=i
              x2=x2+cos(xi*j1/3d0)
20          continue
            hh=x1*(1d0+2d0*x2/3d0)
          elseif(uniform.eq.0.and.apertures.eq.4) then
            do 30 i=1,6
              xi=i
              x2=x2+cos(xi*j1/6d0)
30          continue
            hh=x1*(1d0+x2/2d0)
          elseif(uniform.eq.0.and.apertures.eq.5) then
            do 40 i=1,11
              if(i.eq.10) goto 40
              xi=i
              x2=x2+cos(xi*j1/11d0)
40          continue
            hh=x1*(1d0+2d0*x2/5d0)
          elseif(uniform.eq.0.and.apertures.eq.6) then
            do 50 i=1,17
              if(i.eq.14.or.i.eq.15) goto 50
              xi=i
              x2=x2+cos(xi*j1/17d0)
50          continue
            hh=x1*(1d0+x2/3d0)
        endif
      endif
      return
      end
```

```
c
c
c
      subroutine spectramkr(err,m,retil,f)
      real*8 err(m,m),retil(128),f(128),xii,xi,pi2,sum(128)
      integer p
      pi2=6.283185307
      do 10 i=1,m
       sum(i)=0.0
10    continue
      do 20 i=1,m
       do 30 ii=1,m
        p=abs(i-ii)+1
        sum(p)=sum(p)+err(i,ii)
30     continue
20    continue
      do 50 i=1,128
       xi=real(i)/256
       retil(i)=0
       do 60 ii=1,m
        xii=ii-1
        retil(i)=retil(i)+sum(ii)*cos(pi2*xii*xi)
60     continue
50    continue
      do 80 i=1,128
       f(i)=real(i)/64
       retil(i)=retil(i)/dble(m)
80    continue
      return
      end
```

```
c
c
c             program snrmkr2.f
c
c
      real*4 sum,snr(128),f1(128),lg10,slice(32),count(32)
      real*8 f(128),e(128)
c
      lg10=log(1e1)
c     do 5 j=1,180
c     do 5 j=1,8
      do 5 j=1,32
c       call readfl('minvar2n1024',n,'d',f,'d',e,j,1,128)
c       call readfl('newmtfd005s6',n,'d',f,'d',e,j,1,128)
        call readfl('s5d005u0a6s32',n,'d',f,'d',e,j,1,128)
c       call readfl('s4d005u1a11s32',n,'d',f,'d',e,j,1,128)
c       call readfl('s6d005u1a6s8',n,'d',f,'d',e,j,1,128)
        sum=0
        do 10 i=1,128
          sum=sum+e(i)
          snr(i)=10*log(real(i)/sum)/lg10
          f1(i)=real(i)/64
 10     continue
c       call plotfl('snrminvarn1024',128,'f',f1,'f',snr)
c       call plotfl('snrnwmtfd005s6',128,'f',f1,'f',snr)
        call plotfl('snrs5d005u0a6',128,'f',f1,'f',snr)
c       call plotfl('snrs4d005u1a11',128,'f',f1,'f',snr)
c       slice(j)=snr(64)
c       slice(j)=snr(52)
        slice(j)=snr(49)
        count(j)=j
 5    continue
      call plotfl('slcs5d005u0a6',32,'f',count,'f',slice)
c     call plotfl('slcs4d005u1a11',32,'f',count,'f',slice)
c     call plotfl('slcs6d005u1a6',8,'f',count,'f',slice)
      stop
      end
```

```
c
c
c          program lsqmkr2.f
c
c
           parameter(mm=128)
           real*8 bw(1150,mm),f(128),v(mm,mm),w(mm)
           real*8 g(mm,mm),retil(128),d
           integer*4 i,m,mb,n
           integer*4 max,uniform,apertures
           character dfile*16
c
c          n=1024
c          n=512
           print *, 'Number of pixels in image line?'
           read *, n
           print *, 'snr?'
           read *, snr
           print *, 'd/D?'
           read *, d
           print *, 'uniformity?(yes=1,no=0)'
           read *, uniform
           print *, 'number of apertures?'
           read *, apertures
           print *, 'Largest support?'
           read *, max
           print *, 'Destination File?'
           read *, dfile
           do 5 i=1,max
             m=2*i
             mb=m
             call bwmkr(bw,n,m,d,uniform,apertures)
             call svdcmp(bw,n,m,w,v)
             call gmkr(w,v,m,g)
             call spectramkr(g,m,retil,f,snr,mb)
             call plotfl(dfile,128,'d',f,'d',retil)
 5         continue
           stop
           end
c
c
c
           subroutine bwmkr(bw,n,m,d,uniform,apertures)
           real*8 bw(n,m),hh,d
           integer n,m,uniform,apertures
           do 520 i=1,n
            do 530 ii=1,m
             bw(i,ii)=hh((m-n)/2-ii+i,d,uniform,apertures)
 530        continue
 520       continue
           return
           end
c
c
c
           subroutine gmkr(w,v,m,g)
           real*8 g(m,m),w(m),v(m,m)
c
           do 600 i=1,m
            do 610 ii=1,m
             g(i,ii)=v(i,ii)/w(ii)
 610        continue
 600       continue
           return
           end
c
c
c
```

```
        subroutine spectramkr(err,m,retil,f,snr,mb)
        real*8 err(m,m),retil(128),f(128),xii,xi,pi2,sum(128)
        integer p
        pi2=8d0*atan(1d0)
        do 10 i=1,m
         sum(i)=0d0
 10     continue
        do 20 i=1,m
         do 30 ii=1,m
          p=abs(i-ii)+1
          do 40 iii=1,m
           sum(p)=sum(p)+err(i,iii)*err(ii,iii)
 40       continue
 30      continue
 20     continue
        print *, sum(1)
        do 50 i=1,128
         xi=real(i)/256
         retil(i)=0
         do 60 ii=1,m
          xii=ii-1
          retil(i)=retil(i)+sum(ii)*cos(pi2*xii*xi)
 60      continue
 50     continue
        do 80 i=1,128
         f(i)=real(i)/128
         retil(i)=retil(i)/(snr*mb)
 80     continue
        return
        end
c
c
c
      function hh(j,d,uniform,apertures)
      real*8 hh,d,pi,d1,j1,j2,x1,x2,xj,xk,xi
      integer j,k,l,uniform,apertures
c
      k=apertures
      xk=k
      if(j.eq.0)then
        hh=xk*d/2d0
      else
        pi=4d0*atan(1d0)
        d1=1d0-d
        xj=j
        j1=xj*d1*pi
        j2=xj*d*pi/2d0
        x1=sin(j2)/j2
        x1=x1*x1*d/2d0
        x2=0d0
        if(uniform.eq.1) then
            do 10 i=1,k-1
              xi=i
              x2=x2+(1d0-xi/xk)*cos(xi*j1/(xk-1d0))
 10         continue
            hh=x1*(1d0+2d0*x2)
          elseif(uniform.eq.0.and.apertures.eq.3) then
            do 20 i=1,3
              xi=i
              x2=x2+cos(xi*j1/3d0)
 20         continue
            hh=x1*(1d0+2d0*x2/3d0)
          elseif(uniform.eq.0.and.apertures.eq.4) then
            do 30 i=1,6
              xi=i
              x2=x2+cos(xi*j1/6d0)
 30         continue
            hh=x1*(1d0+x2/2d0)
```

```
           elseif(uniform.eq.0.and.apertures.eq.5) then
             do 40 i=1,11
               if(i.eq.10) goto 40
               xi=i
               x2=x2+cos(xi*j1/11d0)
 40          continue
             hh=x1*(1d0+2d0*x2/5d0)
           elseif(uniform.eq.0.and.apertures.eq.6) then
             do 50 i=1,17
               if(i.eq.14.or.i.eq.15) goto 50
               xi=i
               x2=x2+cos(xi*j1/17d0)
 50          continue
             hh=x1*(1d0+x2/3d0)
         endif
       endif
       return
       end
c
c
c
c        This routine generates the SVD decomposition of an
c        mxn matrix A = UWVtranspose, where U is mxn, and V
c        and W are nxn.  U is column orthogonal, V is row and
c        column orthogonal, and W is diagonal.  U is returned
c        in the array a.
c
c
c
       subroutine svdcmp(a,m,n,w,v)
       implicit real*8 (a-h,o-z)
       parameter (nmax=128)
       dimension a(m,n),w(n),v(n,n),rv1(nmax)
c
       g=0d0
       scale=0d0
       anorm=0d0
       do 25 i=1,n
         l=i+1
         rv1(i)=scale*g
         g=0d0
         s=0d0
         scale=0d0
         if (i.le.m) then
           do 11 k=i,m
             scale=scale+abs(a(k,i))
 11        continue
           if (scale.ne.0d0) then
             do 12 k=i,m
               a(k,i)=a(k,i)/scale
               s=s+a(k,i)*a(k,i)
 12          continue
             f=a(i,i)
             g=-sign(sqrt(s),f)
             h=f*g-s
             a(i,i)=f-g
             if (i.ne.n) then
               do 15 j=l,n
                 s=0d0
                 do 13 k=i,m
                   s=s+a(k,i)*a(k,j)
 13              continue
                 f=s/h
                 do 14 k=i,m
                   a(k,j)=a(k,j)+f*a(k,i)
 14              continue
 15            continue
             endif
```

```
            do 16 k= i,m
              a(k,i)=scale*a(k,i)
16          continue
          endif
        endif
        w(i)=scale *g
        g=0d0
        s=0d0
        scale=0d0
        if ((i.le.m).and.(i.ne.n)) then
          do 17 k=l,n
            scale=scale+abs(a(i,k))
17        continue
          if (scale.ne.0d0) then
            do 18 k=l,n
              a(i,k)=a(i,k)/scale
              s=s+a(i,k)*a(i,k)
18          continue
            f=a(i,l)
            g=-sign(sqrt(s),f)
            h=f*g-s
            a(i,l)=f-g
            do 19 k=l,n
              rv1(k)=a(i,k)/h
19          continue
            if (i.ne.m) then
              do 23 j=l,m
                s=0d0
                do 21 k=l,n
                  s=s+a(j,k)*a(i,k)
21              continue
                do 22 k=l,n
                  a(j,k)=a(j,k)+s*rv1(k)
22              continue
23            continue
            endif
            do 24 k=l,n
              a(i,k)=scale*a(i,k)
24          continue
          endif
        endif
        anorm=max(anorm,(abs(w(i))+abs(rv1(i))))
25    continue
      do 32 i=n,1,-1
        if (i.lt.n) then
          if (g.ne.0d0) then
            do 26 j=l,n
              v(j,i)=(a(i,j)/a(i,l))/g
26          continue
            do 29 j=l,n
              s=0d0
              do 27 k=l,n
                s=s+a(i,k)*v(k,j)
27            continue
              do 28 k=l,n
                v(k,j)=v(k,j)+s*v(k,i)
28            continue
29          continue
          endif
          do 31 j=l,n
            v(i,j)=0d0
            v(j,i)=0d0
31        continue
        endif
        v(i,i)=1d0
        g=rv1(i)
        l=i
32    continue
```

```
      do 39 i=n,1,-1
        l=i+1
        g=w(i)
        if (i.lt.n) then
          do 33 j=l,n
            a(i,j)=0d0
33        continue
        endif
        if (g.ne.0d0) then
          g=1d0/g
          if (i.ne.n) then
            do 36 j=l,n
              s=0d0
              do 34 k=l,m
                s=s+a(k,i)*a(k,j)
34            continue
              f=(s/a(i,i))*g
              do 35 k=i,m
                a(k,j)=a(k,j)+f*a(k,i)
35            continue
36          continue
          endif
          do 37 j=i,m
            a(j,i)=a(j,i)*g
37        continue
        else
          do 38 j= i,m
            a(j,i)=0d0
38        continue
        endif
        a(i,i)=a(i,i)+1d0
39    continue
      do 49 k=n,1,-1
        do 48 its=1,30
          do 41 l=k,1,-1
            nm=l-1
            if ((abs(rv1(l))+anorm).eq.anorm)  go to 2
            if ((abs(w(nm))+anorm).eq.anorm)  go to 1
41        continue
1         c=0d0
          s=1d0
          do 43 i=l,k
            f=s*rv1(i)
            if ((abs(f)+anorm).ne.anorm) then
              g=w(i)
              h=sqrt(f*f+g*g)
              w(i)=h
              h=1d0/h
              c= (g*h)
              s=-(f*h)
              do 42 j=1,m
                y=a(j,nm)
                z=a(j,i)
                a(j,nm)=(y*c)+(z*s)
                a(j,i)=-(y*s)+(z*c)
42            continue
            endif
43        continue
2         z=w(k)
          if (l.eq.k) then
            if (z.lt.0d0) then
              w(k)=-z
              do 44 j=1,n
                v(j,k)=-v(j,k)
44            continue
            endif
            go to 3
          endif
```

```
          if (its.eq.30) pause 'no convergence in 30 iterations'
          x=w(l)
          nm=k-1
          y=w(nm)
          g=rv1(nm)
          h=rv1(k)
          f=((y-z)*(y+z)+(g-h)*(g+h))/(2d0*h*y)
          g=sqrt(f*f+1d0)
          f=((x-z)*(x+z)+h*((y/(f+sign(g,f)))-h))/x
          c=1d0
          s=1d0
          do 47 j=l,nm
            i=j+1
            g=rv1(i)
            y=w(i)
            h=s*g
            g=c*g
            z=sqrt(f*f+h*h)
            rv1(j)=z
            c=f/z
            s=h/z
            f= (x*c)+(g*s)
            g=-(x*s)+(g*c)
            h=y*s
            y=y*c
            do 45 nm=1,n
              x=v(nm,j)
              z=v(nm,i)
              v(nm,j)= (x*c)+(z*s)
              v(nm,i)=-(x*s)+(z*c)
45          continue
            z=sqrt(f*f+h*h)
            w(j)=z
            if (z.ne.0d0) then
              z=1d0/z
              c=f*z
              s=h*z
            endif
            f= (c*g)+(s*y)
            x=-(s*g)+(c*y)
            do 46 nm=1,m
              y=a(nm,j)
              z=a(nm,i)
              a(nm,j)= (y*c)+(z*s)
              a(nm,i)=-(y*s)+(z*c)
46          continue
47        continue
          rv1(l)=0d0
          rv1(k)=f
          w(k)=x
48      continue
3       continue
49    continue
      return
      end
```

```
c
c
c          program snrmkr3.f
c
c
      real*4 sum,snr(128),f1(128),lg10,slice(50),count(50)
      real*8 f(128),e(128)
c
      lg10=log(1e1)
c     do 5 j=1,16
      do 5 j=1,32
c     do 5 j=1,40
c       call readfl('s7d005u1a12s32',n,'d',f,'d',e,j,1,128)
        call readfl('s7d005u1a12n=m',n,'d',f,'d',e,j,1,128)
c       call readfl('s7d005u1a18s40',n,'d',f,'d',e,j,1,128)
c       call readfl('test.lsq',n,'d',f,'d',e,j,1,128)
c       call readfl('grdtstae',n,'d',f,'d',e,j,1,128)
        sum=0
        do 10 i=1,128
          sum=sum+e(i)
          snr(i)=10*log(real(i)/sum)/lg10
          f1(i)=real(i)/128
 10     continue
c       call plotfl('snrs7d005u1a12',128,'f',f1,'f',snr)
        call plotfl('snr7005u112n=m',128,'f',f1,'f',snr)
c       call plotfl('snrs7d005u1a18',128,'f',f1,'f',snr)
c       call plotfl('snrtest.lsq',128,'f',f1,'f',snr)
c       call plotfl('snrgrdtstae',128,'f',f1,'f',snr)
        slice(j)=snr(128)
        count(j)=j
 5    continue
c     call plotfl('slcs7d005u1a12',32,'f',count,'f',slice)
      call plotfl('slc7005u112n=m',32,'f',count,'f',slice)
c     call plotfl('slcs7d005u1a18',40,'f',count,'f',slice)
c     call plotfl('slcstest.lsq',16,'f',count,'f',slice)
c     call plotfl('slcgrdtstae',8,'f',count,'f',slice)
      stop
      end
```

APPENDIX D for Chapter 6

In this appendix we describe the gradient projection algorithm[29] as it was used by us to implement the positivity constraint on the object estimate obtained by unweighted least-squares. We begin with a statement of the problem. Given the observation

$$\mathbf{y} = G\mathbf{x} + \mathbf{n}, \tag{D1}$$

chose $\hat{\mathbf{x}}$ to minimize

$$\epsilon = ||\mathbf{y} - G\hat{\mathbf{x}}||^2, \tag{D2}$$

with the constraining

$$\hat{\mathbf{x}} \geq 0. \tag{D3}$$

We begin by computing the gradient of ϵ of Eq. (D1) with respect to the object estimate $\hat{\mathbf{x}}(n)$ at the n^{th} iteration:

$$\nabla\epsilon(n) = G^T G\hat{\mathbf{x}}(n) - G^T\mathbf{y}. \tag{D4}$$

The search for a solution proceeds along the negative gradient direction unless that direction would violate the constraint. Consider the following iteration equation:

$$\hat{\mathbf{x}}(n+1) = \hat{\mathbf{x}}(n) - \alpha\nabla\epsilon(n), \tag{D5}$$

with

$$\alpha > 0. \tag{D6}$$

If α is chosen so that the above difference equation is stable, then it would converge to the unconstrained solution

$$\hat{\mathbf{x}} = (G^T G)^{-1} G^T \mathbf{y}, \tag{D7}$$

which is the well-known solution to the unconstrained least-squares problem. Now consider how we might modify our algorithm in order to satisfy the positivity constraint. Assume for the moment that all components of $\hat{\mathbf{x}}(n)$ are nonegative. If any component of the gradient is negative, then the corresponding component of $\hat{\mathbf{x}}(n+1)$ will move away from the constraint boundary and there is no problem. If any component of the gradient is strictly positive, and the corresponding component of $\hat{\mathbf{x}}(n)$ is strictly positive, then an appropriate upper bound must be placed on α so that the constraint boundary is not crossed (that component of $\hat{\mathbf{x}}(n+1)$ does not go negative). For the i^{th} component, that bound is obviously $(\hat{\mathbf{x}}(n))_i/(\nabla\epsilon(n))_i$. If any component of the gradient is positive, and the corresponding component of $\hat{\mathbf{x}}(n)$ is zero, then that component of the gradient must be set equal to zero. We are now in a position to define the algorithm. Let $A(n)$ be the set of all indices of $\hat{\mathbf{x}}(n)$ which correspond to zero components. The value of the object estimate at iteration $n+1$ is

$$\hat{\mathbf{x}}(n+1) = \hat{\mathbf{x}}(n) + \alpha(n)\mathbf{s}(n), \tag{D8}$$

where the i^{th} component of $\mathbf{s}(n)$ is given by

$$(s(n))_i = \begin{cases} 0, \ i \in A(n) & \text{and } (\nabla\epsilon(n))_i > 0 \\ -(\nabla\epsilon(n))_i, & \text{else.} \end{cases} \tag{D9}$$

We pick the step size $\alpha(n)$ as follows,

$$\alpha(n) = \min\left[\alpha_1(n), \alpha_2(n)\right], \tag{D10}$$

where

$$\alpha_1(n) = \frac{\mathbf{s}^T(n)\mathbf{s}(n)}{\mathbf{s}^T(n)GG^T\mathbf{s}^T(n)}, \tag{D11}$$

and

$$\alpha_2(n) = \min_{\alpha_i > 0} \left[\alpha_i = \frac{(\hat{\mathbf{x}}(n))_i}{(\nabla\epsilon(n))_i}, i \in A(n) \text{ and } (\nabla\epsilon(n))_i > 0 \right]. \tag{D12}$$

As previously discussed, $\alpha_2(n)$ assures that no constraint boundary is crossed. However, the step size may still be too large, or the set of α_i on Eq. (D12) may be empty. We need an additional bound on step size. A reasonable maximum step size (and the one used in the unconstrained steepest descent algorithm) is one in which $\mathrm{s}(n+1)$ is orthogonal to $\mathrm{s}(n)$. It can be shown that $\alpha_1(n)$ is that step size.

It remains only to choose an initial estimate $\hat{\mathbf{x}}(0)$. We chose the unconstrained solution given by Eq. (D7) with all negative components set equal to zero.

APPENDIX E for Chapter 6

In this appendix we describe the CLEAN[3] algorithm as it was implemented by us to obtain the results of Section 6.5. Let $\mathbf{y}$ be the observation vector given by

$$\mathbf{y} = G\mathbf{x} + \mathbf{n}. \tag{E1}$$

The columns of G are the system point-spread-function shifted and truncated, $\mathbf{x}$ is the object vector, and $\mathbf{n}$ is observation noise. Let $\mathbf{g}_i$ be the i^{th} column of G. In the noiseless case, we can write $\mathbf{y}$ as a linear combination of the $\mathbf{g}_i$ as follows:

$$\mathbf{y} = \sum_i x_i \mathbf{g}_i, \tag{E2}$$

where x_i is the i^{th} component of the object vector $\mathbf{x}$. Given $\mathbf{y}$ and the $\mathbf{g}_i$, CLEAN attempts to iteratively determine the x_i.

Let $\mathbf{y}(n)$ be the n^{th} vector of a sequence of vectors where

$$\mathbf{y}(0) = \mathbf{y}, \tag{E3}$$

and let $y_i(n)$ be the i^{th} component of $\mathbf{y}(n)$. Let $\hat{\mathbf{x}}(n)$ be our estimate of $\mathbf{x}$ after n iterations, with components $\hat{x}_i(n)$, with

$$\hat{x}_i(0) = 0. \tag{E4}$$

Finally, let the integer k_n be the index of the maximum component of $\mathbf{y}(n)$ over the region of support of $\mathbf{x}$, so that $y_{k_n}(n)$ is the maximum value. Then the following set of iteration equations define CLEAN:

$$\mathbf{y}(n+1) = \mathbf{y}(n) - \alpha(n)\mathbf{g}_{k_n}, \tag{E5}$$

$$\hat{x}_i(n+1) = \begin{cases} \hat{x}_i(n) + \alpha(n), & i = k_n \\ \hat{x}_i(n), & i \neq k_n, \end{cases} \tag{E6}$$

where $\alpha(n)$ is given by

$$\alpha(n) = \epsilon \frac{y_{k_n}(n)}{g_{\max}}. \tag{E7}$$

The notation $g_{\max}$ denotes the peak value of $\mathbf{g}_i$, which (presumably) is independent of i. ϵ is a gain parameter—necessarily less than unit, and with a value of $0.4 \leftrightarrow 0.5$ in our work. At each iteration, we also compute

$$\xi(n) = \| \mathbf{y}(n) \|^2 . \tag{E8}$$

If $\xi(n+1) > \xi(n)$ than the iteration is terminated and $\hat{\mathbf{x}}(n)$ is used as the object vector estimate.

APPENDIX F for Chapter 6

Computer Program Listings

This appendix contains the Fortran source code of the computer programs used to generate the data for this chapter.

```
c
c
c          program grdsrch6.f
c
c
           parameter(nn=638,mm=128)
           real*8 bw(nn,mm),h(mm,nn),retilu(128),f(128),x(mm),no(nn)
           real*8 v(mm,mm),w(mm),erru(mm,mm),retilc(128),errc(mm,mm)
           real*8 u(nn,mm)
           real*8 y(nn),gg(mm,mm),d
           integer*4 random,cntr,ni,nj,nt,m,mb,n,nneg,cntr2
           integer*4 uniform,apertures
           character dfile*16
c
           n=512
           print *, 'nt?'
           read *, nt
           print *, 'ni?'
           read *, ni
           print *, 'nj?'
           read *, nj
           print *, 'mb?'
           read *, mb
c          print *, 'm?'
c          read *, m
           print *, 'snr?'
           read *, snr
           print *, 'd/D?'
           read *, d
           print *, 'Uniformity? (yes=1 and no=0)'
           read *, uniform
           print *, 'Number of apertures?'
           read *, apertures
           print *, 'Destination File?'
           read *, dfile
         do 5 m=mb,76,6
           call srandom(29)
           call bwmkr(bw,u,n,m,d,uniform,apertures)
           call ggmkr(bw,n,m,gg)
           call svdmkr(u,n,m,w,v)
           call hmkr(h,u,w,v,m,n)
           do 103 j=1,mm
            do 101 jj=1,mm
             erru(j,jj)=0
             errc(j,jj)=0
101         continue
103        continue
           cntr=0
           cntr2=0
             print *, 'gradsrch trials'
           do 105 iv=1,nt
c            print *,iv
            if (float(iv)/10.0-int(float(iv)/10.0) .lt. 1e-10) then
             print *, iv
            endif
            call randm(x,no,m,n,snr,bw,y,mb)
            call grdsrch(x,h,no,erru,n,m,nt,errc,y,bw,gg,ni,nj,cntr,
     *                   nneg)
            if(nneg.eq.0) then
              cntr2=cntr2+1
```

```
          endif
105      continue
         print *, real(cntr)*real(nj)/real(nt), cntr2
c        call spectramkr(erru,m,retilu,f,snr,mb)
         call spectramkr(errc,m,retilc,f,snr,mb)
c        call plotfl(dfile,128,'d',f,'d',retilu)
         call plotfl(dfile,128,'d',f,'d',retilc)
5     continue
         stop
         end
c
c
c
         subroutine bwmkr(bw,u,n,m,d,uniform,apertures)
         real*8 bw(n,m),u(n,m),hh,d
         integer n,m,uniform,apertures
         do 520 i=1,n
          do 530 ii=1,m
           bw(i,ii)=hh((m-n)/2-ii+i,d,uniform,apertures)
           u(i,ii)=bw(i,ii)
530       continue
520      continue
         return
         end
c
c
c
      function hh(j,d,uniform,apertures)
      real*8 hh,d,pi,d1,j1,j2,x1,x2,xj,xk,xi
      integer j,k,l,uniform,apertures
c
      k=apertures
      xk=k
      if(j.eq.0)then
        hh=xk*d/2d0
      else
        pi=4d0*atan(1d0)
        d1=1d0-d
        xj=j
        j1=xj*d1*pi
        j2=xj*d*pi/2d0
        x1=sin(j2)/j2
        x1=x1*x1*d/2d0
        x2=0d0
        if(uniform.eq.1) then
            do 10 i=1,k-1
              xi=i
              x2=x2+(1d0-xi/xk)*cos(xi*j1/(xk-1d0))
10          continue
            hh=x1*(1d0+2d0*x2)
          elseif(uniform.eq.0.and.apertures.eq.3) then
            do 20 i=1,3
              xi=i
              x2=x2+cos(xi*j1/3d0)
20          continue
            hh=x1*(1d0+2d0*x2/3d0)
          elseif(uniform.eq.0.and.apertures.eq.4) then
            do 30 i=1,6
              xi=i
              x2=x2+cos(xi*j1/6d0)
30          continue
            hh=x1*(1d0+x2/2d0)
          elseif(uniform.eq.0.and.apertures.eq.5) then
            do 40 i=1,11
              if(i.eq.10) goto 40
              xi=i
              x2=x2+cos(xi*j1/11d0)
40          continue
```

```
              hh=x1*(1d0+2d0*x2/5d0)
            elseif(uniform.eq.0.and.apertures.eq.6) then
              do 50 i=1,17
                if(i.eq.14.or.i.eq.15) goto 50
                xi=i
                x2=x2+cos(xi*j1/17d0)
 50           continue
              hh=x1*(1d0+x2/3d0)
          endif
        endif
        return
        end
c
c
c
        subroutine hmkr(h,u,w,v,m,n)
c
        real*8 u(n,m),w(m),v(m,m),h(m,n)
        integer m,n
c
        do 10 i=1,m
          do 20 j=1,n
            h(i,j)=0d0
            do 30 k=1,m
              h(i,j)=h(i,j)+(1/w(k))*v(i,k)*u(j,k)
 30         continue
 20       continue
 10     continue
        return
        end
c
c
c
        subroutine ggmkr(bw,n,m,gg)
c
        real*8 bw(n,m),gg(m,m)
        do 600 i=1,m
         do 610 ii=1,m
          gg(i,ii)=0
          do 620 iii=1,n
           gg(i,ii)=gg(i,ii)+bw(iii,i)*bw(iii,ii)
 620      continue
 610     continue
 600    continue
        return
        end
c
c
c
        subroutine randm(x,no,m,n,snr,bw,y,mb)
c
c         Generates random vectors x, no, and y=BWx+no.  x has mb
c         independent Rayleigh components with variance snr and m-mb
c         zero components.  no has n independent, zero mean, Gaussian
c         components with unit variance.
c
        real*8 u1,u2,u3,s
        real*8 x(m),no(n),bw(n,m),y(n)
        integer*4 MAXINTV,random
        parameter(MAXINTV=2147483647)
        n2=n/2
        do 10 i=1,m
         x(i)=0d0
 10     continue
        do 500 i=1,n2
 3       u1=real(random())/MAXINTV
         if(u1.gt.1.or.u1.eq.0)goto 3
 2       u2=real(random())/MAXINTV
```

```
             if(u2.gt.1.or.u2.eq.0)goto 2
             no(2*i-1)=sqrt(-2*log(u1))*cos(6.2831853*u2)
             no(2*i)=sqrt(-2*log(u1))*sin(6.2831853*u2)
 500        continue
            s=snr/2
            do 505 i=1,mb
 4           u3=real(random())/MAXINTV
             if(u3.gt.1.or.u3.eq.0)goto 4
             x(i+(m-mb)/2)=sqrt(-2*log(u3))*sqrt(s)
 505        continue
            do 510 i=1,n
             y(i)=0
             do 520 ii=1,m
              y(i)=y(i)+bw(i,ii)*x(ii)
 520         continue
              y(i)=y(i)+no(i)
 510        continue
            return
            end
c
c
c
            subroutine grdsrch(x,h,no,erru,n,m,nt,errc,y,bw,gg,
     *                 ni,nj,cntr,nneg)
            include 'fpedefs.f'
            parameter(mm=128)
            real*8 x(m),no(n),erru(m,m),h(m,n),eu(mm),zz,msq,msqold
            real*8 errc(m,m),gg(m,m),grad(mm),dmsq,s(mm)
            real*8 y(n),bw(n,m),z(mm),xg(mm)
            real*8 alpha,beta,ss,sggs,tem1,tem2
            integer cntr,nneg,list(mm),ni,nj,m,n,nt
            nneg=0
            msq=0
c ------------  Compute unconstrained solution and BWy  --------------
            do 700 i=1,m
             xg(i)=0
             z(i)=0
             do 710 ii=1,n
              xg(i)=xg(i)+h(i,ii)*y(ii)
              z(i)=z(i)+bw(ii,i)*y(ii)
 710         continue
             eu(i)=xg(i)-x(i)
             if(xg(i).lt.0) then
              nneg=nneg+1
             endif
 700        continue
            if(nneg.eq.0) goto 200
c ---------------  Gradient Projection Algorithm  -----------------------
c           ------  Disable overflow and divide-by-zero ---
c                   floating point exceptions.
            nabls=fpgetxnabls()
            newabls=and(nabls,not(DIVO))
            newabls=and(newabls,not(OFLOW))
            call fpsetxnabls(newabls)
 100        msqold=msq
            cntr=cntr+1
            do 770 iv=1,nj
             alpha=1.797d+308
             ss=0d0
             sggs=0d0
c        -------  Find active constraints and generate list  --------
             do 705 i=1,m
               list(i)=0
               if(xg(i).le.0d0) then
                 xg(i)=0d0
                 list(i)=1
               endif
 705         continue
```

```
c       --------  Compute gradient and correction vector S  ---------
c                 and pick step size alpha
           do 712 i=1,m
            grad(i)=0d0
            do 713 ii=1,m
              grad(i)=grad(i)+gg(i,ii)*xg(ii)
713         continue
            grad(i)=grad(i)-z(i)
c                  ----  Project gradient onto active,   ----
c                        nonobstructing constraints.
            if(list(i).eq.1.and.grad(i).gt.0d0) then
              s(i)=0d0
            else
              s(i)=-grad(i)
            endif
c       -----  Determine step size alpha:            --------------
c              Determine minimum step size for next constraint,
c              Determine step size for minimum along direction S,
c       -----  Pick smaller of the two       ----------------------
            if(list(i).ne.1) then
              beta=xg(i)/grad(i)
              if(beta.gt.0d0.and.beta.lt.alpha) then
                alpha=beta
              endif
            endif
712        continue
           do 714 i=1,m
            ss=ss+s(i)*s(i)
            do 716 j=1,m
             sggs=sggs+s(i)*gg(i,j)*s(j)
716         continue
714        continue
           if(ss.eq.0d0) goto 300
           beta=ss/sggs
           if(beta.gt.0d0.and.beta.lt.alpha) then
             alpha=beta
c ------------compute new gradient and test for orthogonality ----------
c            with the S direction.
c              zz=0d0
c              do 900 i=1,m
c               grad(i)=0d0
c               do 901 ii=1,m
c                 grad(i)=grad(i)+gg(i,ii)*(xg(ii)+alpha*s(ii))
c901             continue
c               grad(i)=grad(i)-z(i)
c               zz=zz+grad(i)*s(i)
c900            continue
c              print *, 'gTs =', zz
c ------------ End of Test -------------------------------------------
           endif
c       -----------  Compute new Solution   ------
           do 740 i=1,m
             xg(i)=xg(i)+alpha*s(i)
740        continue
770       continue
c ------------------------ Check ||y-Gx|| squared -----------------------
          msq=0
          do 780 i=1,n
           zz=0
           do 790 ii=1,m
            zz=zz+bw(i,ii)*xg(ii)
790        continue
           zz=y(i)-zz
           msq=msq+zz*zz
780       continue
c             print *, msq
          dmsq=abs((msqold-msq)*(10**ni))
          if (dmsq.gt.1e+9) goto 100
```

```
        if (int(dmsq).ne.0) goto 100
c       -----  Enable Floating Point Exceptions  ---
300      call fpsetxflags(0)
         call fpsetxnabls(nabls)
c ---------  Accumulate Error Vector Outer Product and Return  ------------
200     do 720 i=1,m
c           print *, xg(i), grad(i), s(i)
         tem1=xg(i)-x(i)
         do 730 ii=1,m
          tem2=xg(ii)-x(ii)
          erru(i,ii)=erru(i,ii)+eu(i)*eu(ii)/nt
          errc(i,ii)=errc(i,ii)+tem1*tem2/nt
730      continue
720     continue
c       print *, msq, nneg
        return
        end
c
c
c
        subroutine spectramkr(err,m,retil,f,snr,mb)
        real*8 err(m,m),retil(128),f(128),xii,xi,pi2,sum(128)
        integer p
        pi2=8d0*atan(1d0)
        do 10 i=1,m
         sum(i)=0d0
10      continue
        do 20 i=1,m
         do 30 ii=1,m
          p=abs(i-ii)+1
          sum(p)=sum(p)+err(i,ii)
30       continue
20      continue
        print *, sum(1)
        do 50 i=1,128
         xi=real(i)/256
         retil(i)=0
         do 60 ii=1,m
          xii=ii-1
          retil(i)=retil(i)+sum(ii)*cos(pi2*xii*xi)
60       continue
50      continue
        do 80 i=1,128
         f(i)=real(i)/128
         retil(i)=retil(i)/(snr*mb)
80      continue
        return
        end
c
c
c
      subroutine svdmkr(a,m,n,w,v)
c
c        This routine generates the SVD of an mxn matrix A
c        where A = UWVtranspose, with U mxn, and V and W are
c        nxn.  U is column orthogonal, V is row and column
c        orthogonal, and W is diagonal.  U is returned in the
c        array a.  The diagonal of W is returned as the vector w.
c
      implicit real*8 (a-h,o-z)
      parameter (nmax=128)
      dimension a(m,n),w(n),v(n,n),rv1(nmax)
c
      g=0d0
      scale=0d0
      anorm=0d0
      do 25 i=1,n
        l=i+1
```

```
      rv1(i)=scale*g
      g=0d0
      s=0d0
      scale=0d0
      if (i.le.m) then
        do 11 k=i,m
          scale=scale+abs(a(k,i))
 11     continue
        if (scale.ne.0d0) then
          do 12 k=i,m
            a(k,i)=a(k,i)/scale
            s=s+a(k,i)*a(k,i)
 12       continue
          f=a(i,i)
          g=-sign(sqrt(s),f)
          h=f*g-s
          a(i,i)=f-g
          if (i.ne.n) then
            do 15 j=l,n
              s=0d0
              do 13 k=i,m
                s=s+a(k,i)*a(k,j)
 13           continue
              f=s/h
              do 14 k=i,m
                a(k,j)=a(k,j)+f*a(k,i)
 14           continue
 15         continue
          endif
          do 16 k= i,m
            a(k,i)=scale*a(k,i)
 16       continue
        endif
      endif
      w(i)=scale *g
      g=0d0
      s=0d0
      scale=0d0
      if ((i.le.m).and.(i.ne.n)) then
        do 17 k=l,n
          scale=scale+abs(a(i,k))
 17     continue
        if (scale.ne.0d0) then
          do 18 k=l,n
            a(i,k)=a(i,k)/scale
            s=s+a(i,k)*a(i,k)
 18       continue
          f=a(i,l)
          g=-sign(sqrt(s),f)
          h=f*g-s
          a(i,l)=f-g
          do 19 k=l,n
            rv1(k)=a(i,k)/h
 19       continue
          if (i.ne.m) then
            do 23 j=l,m
              s=0d0
              do 21 k=l,n
                s=s+a(j,k)*a(i,k)
 21           continue
              do 22 k=l,n
                a(j,k)=a(j,k)+s*rv1(k)
 22           continue
 23         continue
          endif
          do 24 k=l,n
            a(i,k)=scale*a(i,k)
 24       continue
```

```
            endif
          endif
          anorm=max(anorm,(abs(w(i))+abs(rv1(i))))
25      continue
        do 32 i=n,1,-1
          if (i.lt.n) then
            if (g.ne.0d0) then
              do 26 j=l,n
                v(j,i)=(a(i,j)/a(i,l))/g
26            continue
              do 29 j=l,n
                s=0d0
                do 27 k=l,n
                  s=s+a(i,k)*v(k,j)
27              continue
                do 28 k=l,n
                  v(k,j)=v(k,j)+s*v(k,i)
28              continue
29            continue
            endif
            do 31 j=l,n
              v(i,j)=0d0
              v(j,i)=0d0
31          continue
          endif
          v(i,i)=1d0
          g=rv1(i)
          l=i
32      continue
        do 39 i=n,1,-1
          l=i+1
          g=w(i)
          if (i.lt.n) then
            do 33 j=l,n
              a(i,j)=0d0
33          continue
          endif
          if (g.ne.0d0) then
            g=1d0/g
            if (i.ne.n) then
              do 36 j=l,n
                s=0d0
                do 34 k=l,m
                  s=s+a(k,i)*a(k,j)
34              continue
                f=(s/a(i,i))*g
                do 35 k=i,m
                  a(k,j)=a(k,j)+f*a(k,i)
35              continue
36            continue
            endif
            do 37 j=i,m
              a(j,i)=a(j,i)*g
37          continue
          else
            do 38 j= i,m
              a(j,i)=0d0
38          continue
          endif
          a(i,i)=a(i,i)+1d0
39      continue
        do 49 k=n,1,-1
          do 48 its=1,30
            do 41 l=k,1,-1
              nm=l-1
              if ((abs(rv1(l))+anorm).eq.anorm)  go to 2
              if ((abs(w(nm))+anorm).eq.anorm)  go to 1
41          continue
```

```
1       c=0d0
        s=1d0
        do 43 i=l,k
          f=s*rv1(i)
          if ((abs(f)+anorm).ne.anorm) then
            g=w(i)
            h=sqrt(f*f+g*g)
            w(i)=h
            h=1d0/h
            c= (g*h)
            s=-(f*h)
            do 42 j=1,m
              y=a(j,nm)
              z=a(j,i)
              a(j,nm)=(y*c)+(z*s)
              a(j,i)=-(y*s)+(z*c)
42          continue
          endif
43      continue
2       z=w(k)
        if (l.eq.k) then
          if (z.lt.0d0) then
            w(k)=-z
            do 44 j=1,n
              v(j,k)=-v(j,k)
44          continue
          endif
          go to 3
        endif
        if (its.eq.30) pause 'no convergence in 30 iterations'
        x=w(l)
        nm=k-1
        y=w(nm)
        g=rv1(nm)
        h=rv1(k)
        f=((y-z)*(y+z)+(g-h)*(g+h))/(2d0*h*y)
        g=sqrt(f*f+1d0)
        f=((x-z)*(x+z)+h*((y/(f+sign(g,f)))-h))/x
        c=1d0
        s=1d0
        do 47 j=l,nm
          i=j+1
          g=rv1(i)
          y=w(i)
          h=s*g
          g=c*g
          z=sqrt(f*f+h*h)
          rv1(j)=z
          c=f/z
          s=h/z
          f= (x*c)+(g*s)
          g=-(x*s)+(g*c)
          h=y*s
          y=y*c
          do 45 nm=1,n
            x=v(nm,j)
            z=v(nm,i)
            v(nm,j)= (x*c)+(z*s)
            v(nm,i)=-(x*s)+(z*c)
45        continue
          z=sqrt(f*f+h*h)
          w(j)=z
          if (z.ne.0d0) then
            z=1d0/z
            c=f*z
            s=h*z
          endif
          f= (c*g)+(s*y)
```

```
            x=-(s*g)+(c*y)
            do 46 nm=1,m
              y=a(nm,j)
              z=a(nm,i)
              a(nm,j)= (y*c)+(z*s)
              a(nm,i)=-(y*s)+(z*c)
 46         continue
 47       continue
          rv1(l)=0d0
          rv1(k)=f
          w(k)=x
 48     continue
 3      continue
 49   continue
      return
      end
```

```
c
c
c        program clean4.f
c
c
         parameter(nn=638,mm=128)
         real*8 bw(nn,mm),f(128),x(mm),no(nn)
         real*8 retilc(128),errc(mm,mm)
         real*8 y(nn),d,eps,psf(2*nn)
         integer*4 random,cntr,nt,m,mb,n
         integer*4 uniform,apertures,flag
         character dfile*16
c
c        n=512
         print *, 'nt?'
         read *, nt
         print *, 'eps?'
         read *, eps
c        print *, 'mb?'
c        read *, mb
c        print *, 'm?'
c        read *, m
         print *, 'snr?'
         read *, snr
         print *, 'Add noise? (yes=1 and no=0)'
         read *, flag
         print *, 'd/D?'
         read *, d
         print *, 'Uniformity? (yes=1 and no=0)'
         read *, uniform
         print *, 'Number of apertures?'
         read *, apertures
         print *, 'Destination File?'
         read *, dfile
     do 10 i=1,23
       if(i.lt.21) then
        m=2*i
       else
        m=(i-16)*10
       endif
        mb=m
        n=m
         call srandom(29)
         call bwmkr(bw,psf,n,m,d,uniform,apertures)
         do 103 j=1,mm
          do 101 jj=1,mm
           errc(j,jj)=0
101       continue
103      continue
         cntr=0
           print *, 'clean trials'
         do 105 iv=1,nt
c          print *,iv
          if (float(iv)/10.0-int(float(iv)/10.0) .lt. 1e-10) then
           print *, iv
          endif
          call randm(x,no,m,n,snr,bw,y,mb,flag)
          call clean(x,n,m,nt,eps,errc,y,cntr,d,apertures,psf)
105      continue
         print *, real(cntr)/real(nt)
         call spectramkr(errc,m,retilc,f,snr,mb)
         call plotfl(dfile,128,'d',f,'d',retilc)
10   continue
         stop
         end
c
c
c
```

```
              subroutine bwmkr(bw,psf,n,m,d,uniform,apertures)
              real*8 bw(n,m),psf(2*n),hh,d
              integer k,n,m,uniform,apertures
              do 500 i=1,2*n
               psf(i)=hh(i-n,d,uniform,apertures)
 500          continue
              do 520 i=1,n
               do 530 ii=1,m
                k=(m-n)/2-ii+i+n
                bw(i,ii)=psf(k)
 530           continue
 520          continue
              return
              end
c
c
c
      function hh(j,d,uniform,apertures)
      real*8 hh,d,pi,d1,j1,j2,x1,x2,xj,xk,xi
      integer j,k,l,uniform,apertures
c
      k=apertures
      xk=k
      if(j.eq.0)then
        hh=xk*d/2d0
      else
        pi=4d0*atan(1d0)
        d1=1d0-d
        xj=j
        j1=xj*d1*pi
        j2=xj*d*pi/2d0
        x1=sin(j2)/j2
        x1=x1*x1*d/2d0
        x2=0d0
        if(uniform.eq.1) then
            do 10 i=1,k-1
              xi=i
              x2=x2+(1d0-xi/xk)*cos(xi*j1/(xk-1d0))
 10         continue
            hh=x1*(1d0+2d0*x2)
          elseif(uniform.eq.0.and.apertures.eq.3) then
            do 20 i=1,3
              xi=i
              x2=x2+cos(xi*j1/3d0)
 20         continue
            hh=x1*(1d0+2d0*x2/3d0)
          elseif(uniform.eq.0.and.apertures.eq.4) then
            do 30 i=1,6
              xi=i
              x2=x2+cos(xi*j1/6d0)
 30         continue
            hh=x1*(1d0+x2/2d0)
          elseif(uniform.eq.0.and.apertures.eq.5) then
            do 40 i=1,11
              if(i.eq.10) goto 40
              xi=i
              x2=x2+cos(xi*j1/11d0)
 40         continue
            hh=x1*(1d0+2d0*x2/5d0)
          elseif(uniform.eq.0.and.apertures.eq.6) then
            do 50 i=1,17
              if(i.eq.14.or.i.eq.15) goto 50
              xi=i
              x2=x2+cos(xi*j1/17d0)
 50         continue
            hh=x1*(1d0+x2/3d0)
        endif
      endif
```

```
      return
      end
c
c
c
         subroutine randm(x,no,m,n,snr,bw,y,mb,flag)
c
c          Generates random vectors x, no, and y=BWx+no.  x has mb
c          independent Rayleigh components with variance snr and m-mb
c          zero components.  no has n independent, zero mean, Gaussian
c          components with unit variance.
c
         real*8 u1,u2,u3,s
         real*8 x(m),no(n),bw(n,m),y(n)
         integer*4 MAXINTV,random,flag
         parameter(MAXINTV=2147483647)
         n2=n/2
         do 10 i=1,m
          x(i)=0d0
10       continue
         do 500 i=1,n2
3         u1=real(random())/MAXINTV
          if(u1.gt.1.or.u1.eq.0)goto 3
2         u2=real(random())/MAXINTV
          if(u2.gt.1.or.u2.eq.0)goto 2
          no(2*i-1)=sqrt(-2*log(u1))*cos(6.2831853*u2)
          no(2*i)=sqrt(-2*log(u1))*sin(6.2831853*u2)
500      continue
         s=snr/2
         do 505 i=1,mb
4         u3=real(random())/MAXINTV
          if(u3.gt.1.or.u3.eq.0)goto 4
          x(i+(m-mb)/2)=sqrt(-2*log(u3))*sqrt(s)
505      continue
         do 510 i=1,n
          y(i)=0
          do 520 ii=1,m
           y(i)=y(i)+bw(i,ii)*x(ii)
520       continue
          if(flag.eq.1) then
           y(i)=y(i)+no(i)
          endif
510      continue
         return
         end
c
c
c
        subroutine clean(x,n,m,nt,eps,errc,y,cntr,d,apertures,psf)
c
         parameter(mm=128)
         real*8 x(m),y(n),errc(m,m),xhat(mm),alpha,xa,sumsq,ssold
         real*8 tmp,h0,eps,psf(2*n),d
         integer*4 cntr,n,m,k,nt,apertures
c
         sumsq=1d37
         xa=apertures
         h0=xa*d/2d0
         do 10 i=1,mm
           xhat(i)=0d0
10       continue
100      ssold=sumsq
         cntr=cntr+1
         tmp=y((n-m)/2+1)
         k=((n-m)/2+1)
         do 20 i=(n-m)/2+2,(n+m)/2
           if(y(i).gt.tmp) then
             tmp=y(i)
```

```
              k=i
            endif
 20       continue
          alpha=eps*tmp/h0
          do 30 i=1,n
            y(i)=y(i)-alpha*psf(i-k+n)
 30       continue
          xhat(k-(n-m)/2)=xhat(k-(n-m)/2)+alpha
          sumsq=0d0
          do 40 i=1,n
            sumsq=sumsq+y(i)*y(i)
 40       continue
          if(sumsq.lt.ssold) goto 100
          print *, ssold, sumsq
c  ---------  Accumulate Error Vector Outer Product and Return  ------------
          do 50 i=1,m
           tem1=x(i)-xhat(i)
           do 60 ii=1,m
            tem2=x(ii)-xhat(ii)
            errc(i,ii)=errc(i,ii)+tem1*tem2/nt
 60        continue
 50       continue
          return
          end

c
c
c
          subroutine spectramkr(err,m,retil,f,snr,mb)
          real*8 err(m,m),retil(128),f(128),xii,xi,pi2,sum(128)
          integer p
          pi2=8d0*atan(1d0)
          do 10 i=1,m
           sum(i)=0d0
 10       continue
          do 20 i=1,m
           do 30 ii=1,m
            p=abs(i-ii)+1
            sum(p)=sum(p)+err(i,ii)
 30        continue
 20       continue
          print *, sum(1)
          do 50 i=1,128
           xi=real(i)/256
           retil(i)=0
           do 60 ii=1,m
            xii=ii-1
            retil(i)=retil(i)+sum(ii)*cos(pi2*xii*xi)
 60        continue
 50       continue
          do 80 i=1,128
           f(i)=real(i)/128
           retil(i)=retil(i)/(snr*mb)
 80       continue
          return
          end
```

```
c
c
c            program snrmkr2.f
c
c
      real*4 sum,snr(128),f1(128),lg10,slice(50),count(50)
      real*8 f(128),e(128)
c
      lg10=log(1e1)
c     do 5 j=1,16
      do 5 j=1,32
c     do 5 j=1,40
c       call readfl('s7d005u1a12s32',n,'d',f,'d',e,j,1,128)
        call readfl('s7d005u1a12n=m',n,'d',f,'d',e,j,1,128)
c       call readfl('s7d005u1a18s40',n,'d',f,'d',e,j,1,128)
c       call readfl('test.lsq',n,'d',f,'d',e,j,1,128)
c       call readfl('grdtstae',n,'d',f,'d',e,j,1,128)
        sum=0
        do 10 i=1,128
          sum=sum+e(i)
          snr(i)=10*log(real(i)/sum)/lg10
          f1(i)=real(i)/128
10      continue
c       call plotfl('snrs7d005u1a12',128,'f',f1,'f',snr)
        call plotfl('snr7005u112n=m',128,'f',f1,'f',snr)
c       call plotfl('snrs7d005u1a18',128,'f',f1,'f',snr)
c       call plotfl('snrtest.lsq',128,'f',f1,'f',snr)
c       call plotfl('snrgrdtstae',128,'f',f1,'f',snr)
        slice(j)=snr(128)
        count(j)=j
5     continue
c     call plotfl('slcs7d005u1a12',32,'f',count,'f',slice)
      call plotfl('slc7005u112n=m',32,'f',count,'f',slice)
c     call plotfl('slcs7d005u1a18',40,'f',count,'f',slice)
c     call plotfl('slcstest.lsq',16,'f',count,'f',slice)
c     call plotfl('slcgrdtstae',8,'f',count,'f',slice)
      stop
      end
```